Développement d'Applications Qt

version 1.02 – M.Grimaldi

mis à jour le : 24 novembre 2019

Table des matières

I.Introduction....2
II.Installation du framework Qt....2
III.Organisation interne du framework Qt....3
IV.Applications de base - apprentissage....4
A.Application console utilisant les classes de Qt et développées par l'utilisateur....4
i.La classe QDate....4
ii.La classe QTime....5
iii.Classes QPolygon et QPoint....5
iv.Classe QFile pour générer des fichiers CSV....6
v.Classe QFile avec un flux QTextStream....7
vi.Ecriture et lecture de fichiers binaires....7
vii.Classe QImage pour manipuler des images....7
viii.Classe QDir pour accéder aux répertoires du système de fichier....8
B.Application GUI....9
i.Placement manuel de composants sur un QWidget....9
ii.Application utilisant QtDesigner....9
iii.Les dialogues standards....10
iv.Gestion évoluée de l'interface - Layout....10
v.Mise en ressource d'images....11
vi.Dessin sur l'interface et événements souris....12
V.Application plus évoluées utilisant des classes Qt remarquables....14
i.Paramètres non volatiles, QSettings....14
ii.Mise en œuvre de timer, QTimer....14
iii.Génération d'impression ou de document pdf....15
iv.Applications *multithread*, Qthread....15
v.Applications en réseau *broadcast et capture UDP*....16
vi.Application client/serveur TCP....17
VI.Application utilisant des bibliothèques externes....17
i.Utilisation de la bibliothèque libsensors....18
ii.Représentation graphique de données avec QWT....19
iii.Transformée de Fourier avec libfftw....20
iv.Traitement d'image avec Opencv....21
VII.Conclusion....22

I. Introduction

Le framework Qt est très largement employé[1] dans les applications informatiques standard de type *Desktop* mais également dans les applications embarquées sur plate-forme mobiles *Android*, *IOS* ou autre[2].

Par rapport à d'autres frameworks, Qt présente l'avantage d'être *encore* libre et open source, de s'exécuter aussi bien sous Windows, Linux et MasOS, et surtout d'avoir des fonctionnalités de cross compilation directement intégrées à son outil de développement QtCreator.

Dans ce module, nous allons voir les principales fonctionnalités du framework Qt, sachant que la richesse de ces dernières demanderait plusieurs années de travail pour en faire le tour.

Pour apprendre d'autrui, il faut se reconnaître dans l'autre et lui ressembler. C'est ce que montrent des chercheurs qui étudient les mécanismes cérébraux et comportementaux de l'apprentissage social. Ils tentent notamment d'identifier les facteurs qui y sont favorables ou qui, au contraire, perturbent l'acquisition des connaissances. Ce document est donc basé sur le principe: « *regardez comment je fais, faites le vous même puis tirez en la logique* ».

Deux icônes apparaissent dans au long de ce document :

« Regardez comment je fais »

1. chargez et exécutez le programme que vous trouverez dans l'archive:
   http://grimaldi.univ-tln.fr/files/developpementApplicationsQt.zip
2. analysez le **pour en comprendre le principe**, et non pour vérifier si il marche !

À vous de travailler !

Tous les travaux demandés sur lesquels figurera l'icône  , seront rendus au fil des séances par email, sous la forme d'un dossier compressé, archive, ne contenant que les codes source (sans exécutable) à: **grimaldi@univ-tln.fr**

Le nom de l'archive contiendra votre nom et le titre du programme, séparés par des '-'

ex: Grimaldi-fichier-texte-csv.zip

II. Évaluation

ce module d'enseignement devant faire l'objet d'une évaluation comme il se doit, celle-ci portera sur trois points:

1 Pour découvrir Qt et ses applications phares : http://qt.developpez.com/faq/?page=intro

2 Android, iOS, WinRT (incluant Windows Phone), Sailfish OS, Tizen.

1. pour 20%, une note d'investissement au cours des séances, à ma discrétion personnelle, portant sur votre investissement, votre autonomie, votre capacité à rester concentré sur un programme malgré les difficultés, votre capacité à utiliser l'aide de Qt, l'internet, et tout ce qui est mis à votre disposition pour trouver la solution.
2. pour 80%, une note portant sur le projet individuel de jeu de Poker en ligne.

III. Installation du framework Qt

Le framework est disponible à l'URL https://www.qt.io/download/ sous deux versions:

« Open source », Disponible sous licences GPL et LGPLv3. Lors de l'élaboration d'une application avec Qt Open Source, vous devez vérifier que vous êtes en accord avec les termes de la licence GPL ou LGPL et de vos obligations légales. (lien : www1.qt.io/download-open-source/)

« **Commercial** », Licence commerciale sans obligations GPL / LGPL, offrant les services du support Officiel Qt, une licence à durée déterminée ou perpétuelle, des relations directes avec The Qt Company avec accès prioritaire à Qt R & D.

pour installer Qt derrière un proxy, je vous conseille de prendre l'installation *offline*

Nota : Sous Windows, il existe plusieurs version de Qt en fonction de la chaîne de compilation que vous utilisez : visual studio 2013 ou 2015, ou MinGW.

- Windows 64-bit (VS 2015, 918 MB)
- Windows 32-bit (VS 2015, 924 MB)
- Windows 64-bit (VS 2013, 904 MB)
- Windows 32-bit (VS 2013, 909 MB)
- Windows 32-bit (MinGW 5.3.0, 1.1 GB)
-

Sous Linux :

- Linux 64-bit (715 MB)
- Linux 32-bits, uniquement en version *online*.

À vous de travailler ! - Premier travail...

Dés que vous rentrerez chez vous, installer un version de Qt et QtCreator sur votre machine personnelle.

IV. Organisation interne du framework Qt

Qt est une bibliothèque logicielle **orientée objet** et développée en **C++**. A l'origine, ses principales fonctionnalités tournaient autour des composants d'interface graphique appelés "widgets". De nombreuses fonctionnalités se sont greffées à cela pour parfaire la portabilité du code comme la gestion de l'accès aux base de données, les connexions réseaux, des facilités pour la programmation concurrente, ...

Qt **permet la portabilité des applications** qui n'utilisent que ses composants par simple **recompilation** du code source. Les systèmes d'exploitations supportés sont nombreux : Linux, Windows, MacOs. On peut également procéder à un *cross compilation* pour développer sur des systèmes embarqués comme Android, ou IOS.

Figure 1 - Organisation interne du framework Qt

V. Applications de base - apprentissage

A. Application console utilisant les classes de Qt et développées par l'utilisateur

i. La classe QDate

Supposons que l'on vous demande de réaliser un programme permettant de :

1. Entrer votre date de naissance sous la forme jj/mm/aaaa
2. Déterminer quel était son jour de la semaine (lundi, mardi, .etc.)
3. Déterminer quel était son jour de l'année ?
4. Déterminer combien de jours la séparent d'aujourd'hui ?
5. 270 jours avant, quel était la date et le jour de la semaine ?
6. Combien de jours avait cette année là (était elle bissextile) ?

Bien sûr vous pouvez trouver les algorithmes qui permettent d'effectuer ces calculs, et les programmer vous même, mais les développeurs de Qt l'on déjà fait pour vous, au sein d'une classe **QDate**.

À vous de travailler !

Écrire ce programme, pour cela, aller dans l'aide de Qt et étudier la classe QDate, notamment:

ses constructeurs, les méthodes currentDate(), toString(), dayOfYear(), year(), longDayName(), dayOfWeek(),daysTo(), addDays(), daysInYear()

Vous aurez besoin d'utiliser des chaînes caractères, classe QString. Vous pouvez également consulter l'aide en ligne à ce sujet.

Exemple de résultat attendu:

```
votre date de naissance ? :27/03/1995
Aujourd'hui: 28.11.2017, 332 ème jour de l'année 2017
Vous êtes né(e) le: 27.03.1995, un lundi, il y a -8282 jours/aujourd'hui
270 jours avant: 30.06.1994, c'était un jeudi, il y a -8552 jours/aujourd'hui,
c'était en 1994 une année de 365 jours
```

Pour voir la correction, **pas tout de suite**:

ouvrir le projet dans : **mise-en-oeuvre-qdate-console**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

ii. La classe QTime

combien temps est nécessaire à votre machine pour exécuter la fonction rand (un nombre aléatoire) en c++

À vous de travailler !

Aller dans l'aide de Qt et étudier la classe QTime

ses constructeurs, les méthodes start(), restart, elapsed()

Vous aurez besoin d'appeler plusieurs fois la fonction **rand** pour pouvoir mesurer un temps conséquent.

Exemple de résultat attendu:

```
la fonction rand prend 6.5 ns
```

Pour voir la correction, **pas tout de suite**:

ouvrir le projet dans : **un-chronometre-avec-qtime**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

iii. Classes QPolygon et QPoint

Sujet : Nous voudrions traiter des polygones du plan, c'est à dire

- les définir à partir de leurs sommets et d'un nom (une chaîne de caractères)
- les translater d'un vecteur donné
- les afficher sur la console sous la forme

```
P1 = (0,0), (0,10), (10,10), (0,0),
```

- calculer leur aire
- calculer leur intersection avec un autre polygone
- vérifier si un point donné fait partie du polygone ou non

exemple : P2, en bleu, est le translaté (5,5) de P1, en rouge. Pi est l'intersection de P1 et de P2

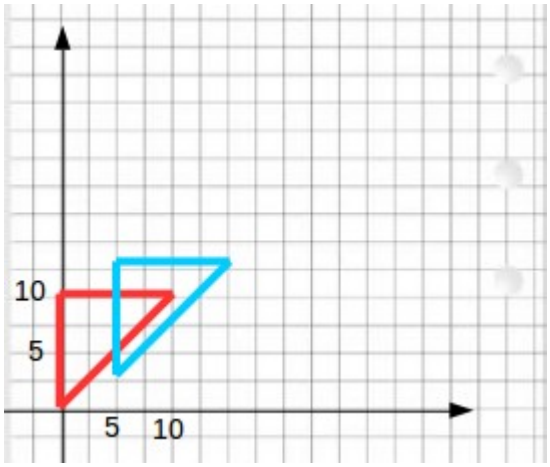


Figure 2 - polygones, exemple 1

Exemple de résultat attendu:

```
P1 = (0,0), (0,10), (10,10), (0,0),
P2 = (5,2), (5,12), (15,12), (5,2),
PI = (5,10), (10,10), (5,5), (5,10),
aire: p1=50 p2=50 pi=12.5
entrez un point
x ? =10
y ? =10
pi contient le point donné
```

Pour voir la correction, **pas tout de suite**:

ouvrir le projet dans : **classe-heritee-poygon-console**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

iv. Classe QFile pour générer des fichiers CSV

Il est parfois bien utile de sauver des données dans un fichier pour les réutiliser plus tard. Dans Qt, il existe une classe **QFile** qui encapsule toutes les fonctionnalités liées à la gestion de ficher (texte ou binaire).

Dans l'exemple, nous écrivons les échantillons d'une sinusoïde dans un fichier CSV (*comma separated values*), puis nous les relisons pour les imprimer sur l'écran.

Vous pouvez vérifier le contenu du fichier avec **LibreOffice Calc** et même en tracer la courbe.

ouvrir le projet dans : **fichier-texte-csv**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Modifiez le programme pour pouvoir entrer la fréquence, l'amplitude et la durée de la sinusoïde en paramètres de la ligne de commande

exemple de ligne de commande :

```
fichier-texte –frequency=200 –amplitude=20 -duration=1000
```

Visualisez le fichier résultant avec LibreOffice Calc

v. Classe QFile avec un flux QTextStream

Pour faire exactement la même chose que précédemment, on peut également utiliser un flux texte, **QTextStream**, pour avoir accès aux redirections << et >>.

ouvrir le projet dans : **fichier-texte-csv-flux**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

vi. Écriture et lecture de fichiers binaires

Quand on a beaucoup de données, il est préférable d'utiliser des fichiers binaires plutôt que des fichiers texte. (1 octet de donnée = 1 octet sauvé)

La classe **QFile** permet également de gérer ce format de données.

ouvrir le projet dans : **fichier-binaire**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

Nota : vous pouvez visualiser un fichier binaire en l'ouvrant sous QtCreator.

À vous de travailler !

Écrire un programme qui écrit les 10 nombres entiers suivants : 3, 16, 64, 65, 127, 156, 48, 1540, -1, -54 dans un fichier binaire nommé **value.bin** (entiers signés sur 16 bits).

Éditer le fichier en hexadécimal au moyen de QtCreator pour corriger les valeurs 64 en 65, 156 en 150, et -54 en +54.

Vérifier en relisant le fichier que ces modifications ont bien été effectuées.

vii. Classe QImage pour manipuler des images

Si on vous demandait d'écrire un programme qui :

1. Ouvre une image à partir d'un fichier jpeg ou autre (bmp, png, .etc.).
2. Échange les plans rouge et vert puis sauve le résultats
3. Transforme l'image en N&B et sauve le résultats
4. Remplace le fond noir de l'image précédente par du vert et sauve le résultats

bien entendu, vous n'avez que 10 mn pour faire ça. Vous seriez bien embêté !

Avec la classe **QImage**, les développeurs de Qt ont prévu toutes les actions que l'on peut imaginer faire sur des images : enregistrer, charger, accéder aux pixels, convertir les formats, .etc.

ouvrir le projet dans : **classe-qimage-console**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Modifier le programme donné pour calculer l'histogramme d'une image en niveaux de gris (si elle est en couleur, la convertir en niveaux de gris préalablement), puis le sauver dans un fichier CSV.

Tracer cet histogramme dans LibreOffice Calc (ou Microsoft Excel si vous préférez payer pour quelque chose que vous pouvez avoir gratuitement ...)

viii. Classe QDir pour accéder aux répertoires du système de fichier

Et si on demandait maintenant de convertir toutes les images JPEG d'un répertoire source, spécifié en 1er argument de la ligne de commande, en images BMP dans un répertoire destination, spécifié en 2ème argument de la ligne de commande.

Si le répertoire source n'est pas spécifié, le programme affiche un texte d'aide

Si le répertoire destination n'est pas spécifié, il sauve les images dans un répertoire nommé **converted** sous le répertoire destination.

Nota : Les images JPEG peuvent avoir l'extension .jpg, .jpeg, .JPG ou .JPEG.

ouvrir le projet dans : **convertisseur-images-repertoires**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Écrire un programme qui demande à l'utilisateur d'entrer un nom de répertoire, vérifier s'il existe dans le *home directory* de l'utilisateur linux, et s'il existe, le vider, sinon le créer.

Utiliser la fonction *getenv*() du c++ pour récupérer la variable d'environnement HOME, exemple: *getenv*("HOME") et la méthode *removeRecursively*() de la classe **QDir**,

B. Application GUI

Jusqu'à présent, nous nous sommes contentés d'écrire des applications « *console* » comme on en faisait dans les années 80 (ça doit être dû à l'age du prof ...).

Nous allons maintenant mettre en œuvre des applications graphiques présentant une interface dans une fenêtre graphique. C'est quand même plus joli !

i. Placement manuel de composants sur un QWidget

La classe **QWidget** encapsule toutes les fonctionnalités d'un composant visible. Cette classe est, entre autre, la classe mère[3] de tous les autres composants (QPushButton, QLineEdit, QSpinBox, QMenu, *.etc*.)

Si on instancie un **QWidget** avec son constructeur par défaut et qu'on l'affiche (méthode **show**), on obtient une fenêtre graphique vide, mais qui a déjà tous les comportement des fenêtres graphiques (réduction, redimensionnement, déplacement, *.etc*.)

Il est possible d'insérer d'autres composants sur cette interface pour réaliser une application. Vous découvrirez au passage une minuscule échantillon de composants graphiques et leur comportements respectifs et une première introduction au processus de SIGNAL/SLOT.

ouvrir le projet dans : **widget-placement-manuel-composants**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

ii. Application utilisant QtDesigner

Comme vous avez pu le remarquer dans l'exemple précédent, il n'est pas très commode de placer des composants sur l'interface par programme ni de les utiliser. Pourtant, savoir le faire est primordiale quand l'interface n'est pas toujours la même, mais qu'elle dépend du déroulement du programme.

Quand l'interface est fixée à l'avance, les développeurs de Qt ont envisagé une autre solution consistant à utiliser un *designer*, *QtDesigner*, pour décrire l'interface de façon interactive, « avec la souris », ainsi que son comportement : connexion de signaux à des *slots*.

Cette interface est alors entièrement décrite dans un fichier au format XML.

Lors de la construction de l'application, *QtCreator* se charge de générer tout le code source

3 Classe fille, classe mère, héritage. Si vous ne connaissez pas demander au prof de vous le rappeler (ou expliquer) !

nécessaire pour construire l'interface, telle que vous l'avez décrite avec le *Designer*. (*super*!)

ouvrir le projet dans : **application-widget-qt-designer**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Créer une interface permettant de saisir une fiche de renseignement sur une personne comportant, le nom, le prénom (**QLineEdit**), la date de naissance (**QDateEdit**), le sexe (**QComboBox**), le numéro de sécurité social (13 chiffres préformaté avec le sexe, l'année, et le mois), l'adresse sur plusieurs lignes (**QTextEdit**), le numéro de téléphone (**QLineEdit** formaté sur 10 chiffres avec des espaces).

Un bouton de validation permettra de sauver cette P**ersonne** dans un fichier au format CSV.

iii. Les dialogues standards

Le framework Qt possède une multitude de boites de dialogue prédéfinis couvrant la majorité des opérations d'échange avec l'utilisateur : saisie d'un nombre, d'une chaîne de caractère, question, alerte, navigation dans le système de fichier, .etc.

ouvrir le projet dans : **tous-les-dialogues**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Créez une application graphique qui permet d'ouvrir une image existante au moyen d'un dialogue d'ouverture de fichier, de la charger dans une **QImage**, de calculer son histogramme comme cela a été fait au paragraphe vii à la page 7, et enfin de sauver l'histogramme sous la forme d'un fichier CSV, à un autre emplacement choisi par un dialogue de choix de répertoire.

iv. Gestion évoluée de l'interface - Layout

Jusqu'à maintenant, la taille et la disposition des composants que l'on a placé sur l'interface ne dépendait pas de la taille de la fenêtre principale. Cela peut être très ennuyeux si on exécute le programme sur des machines différentes, n'ayant pas la même résolution et disposition d'écran ; exemple : les Desktops ont un écran plus large que haut, les smartphones c'est l'inverse.

Qt dispose d'en ensemble de conteneurs spécifiques, le **QLayout**, permettant de pallier à ce problème.

Sur le Designer, on les retrouve dans la barre latérale :

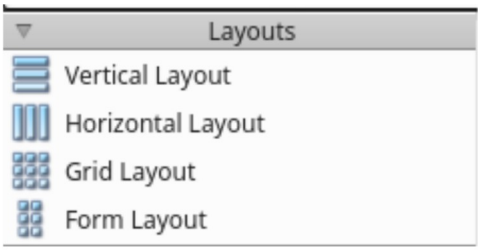


Figure 3 - différents types de *Layouts*

ouvrir le projet dans : **Utilisation-des-Layout-pour-construire-l-interface**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

Le programme ne trouve pas les images, je sais ! Pourquoi ?

À vous de travailler !

Modifier le programme précédent pour que l'on puisse utiliser une base d'images personnalisée, stockée dans un répertoire. Le menu principal devra avoir une option « choisir la base d'images » permettant de choisir le répertoire au moyen d'un dialogue de Qt.

v. Mise en ressource d'images

Pour régler le problème précédent (nécessité de mettre les images dans le répertoire d'exécution), une solution consiste à intégrer les images à l'exécutable par le biais des ressources. On peut mettre dans les ressources toute donnée , quelque soit son type : une image, un son, un fichier texte, un fichier binaire, .etc.

Créer un fichier de ressources :

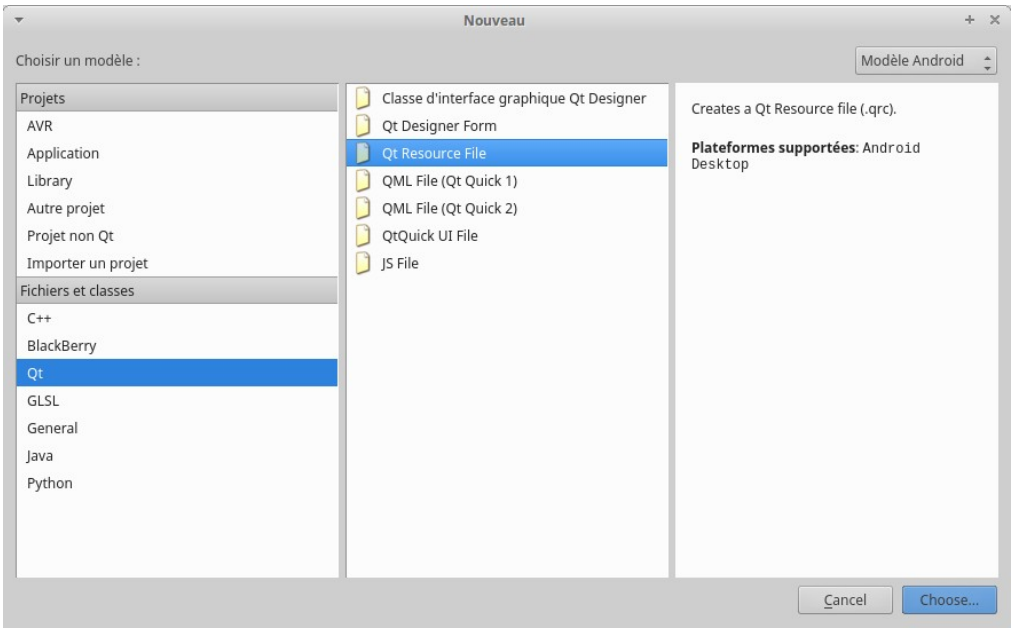


Figure 4 - Insérer les images dans le fichier de ressource.

Pour accéder à une ressource dans le programme, il faut utiliser le nom de son fichier dans la ressource précédé par la chaîne ":". Exemple : si l'image est dans la racine, **":/image.jpg"**.

ouvrir le projet dans : **Utilisation-des-ressources-pour-accéder-à-des-images**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

vi. Dessin sur l'interface et événements souris

Pour dessiner directement sur l'interface d'un **QWidget** (fenêtre graphique), il faut redéfinir sa méthode *paintEvent* comme suit, dans la déclaration de la classe (*header*) :

```
private slots:
        void paintEvent(QPaintEvent *event);
```

Dans l'implémentation de la classe, il suffit d'instancier un objet **QPainter** en lui passant le pointeur **this** (moi, le widget sur lequel on veut dessiner) comme argument :

```
void Widget::paintEvent(QPaintEvent *event)
{
  QPainter painter(this);
```

L'objet **painter** contient alors tout un ensemble de méthodes pour dessiner les lignes, des cercles, des images, changer de crayon (pen), de couleur, *.etc*.

Pour récupérer les événements de la souris sur un **QWidget** (fenêtre graphique), il faut redéfinir sa méthode *mousePressEvent* comme suit, dans la déclaration de la classe (*header*) :

```
private slots:
        void mousePressEvent ( QMouseEvent * event );
```

L'argument ***event*** de la fonction, du type QMouseEvent *, contient toutes les méthodes permettant de récupérer les événements souris : position du click, de la roulette, état du clavier (*ctr*, *shift*, *alt*, *.etc*.) lors du click.

ouvrir le projet dans : **dessins-sur-l'interface-et-gestion-souris**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

À vous de travailler !

Écrire un programme qui affiche l'image d'un feu rouge, comme le montre la figure 5,ci-dessous, quand on clique sur un des trois feux, il s'éclaire. Pour cela, vous devrez utiliser une image que vous pouvez *copier/coller* à partir de ce document, sur laquelle vous dessinerez un cercle plein de la couleur correspondante à la partie à allumer. (notation *html* des couleurs à utiliser : rouge=f4162b, orange=f0a83e, vert= 5bf03e, diamètre du cercle ~ 70 pixels - utiliser un logiciel de dessin pour trouver les positionnements)

Figure 5 - Feux rouge qui réagit aux cliques

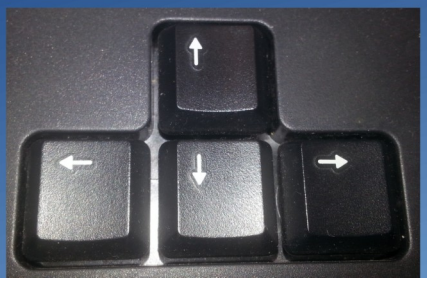
Pour récupérer les événements du clavier sur un **QWidget** (fenêtre graphique), il faut redéfinir sa méthode *keyPressEvent* comme suit, dans la déclaration de la classe (*header*) :

```
private slots:
        void keyPressEvent(QKeyEvent * event);
```

Un peu comme nous l'avons fait pour la souris, et le dessin.

À vous de travailler !

Écrire un programme qui dessine un cercle au centre de la fenêtre puis le déplace à *droite*, à *gauche*, en *haut* ou en *bas* avec les touches clavier correspondantes.



VI. Application plus évoluées utilisant des classes Qt remarquables

i. Paramètres non volatiles, QSettings

Il est parfois utile qu'un programme garde en mémoire des valeurs de certains de ses paramètres comme un identifiant, un mot de passe, un chemin, une adresse IP, *.etc.*, d'une exécution à l'autre.

La première solution à ce problème est de les enregistrer dans un fichier, texte ou binaire, et de les relire à chaque démarrage du programme. Maintenant que vous commencez à bien connaître Qt, vous imaginez bien que les développeurs ont prévu cette situation.

La classe **QSettings** permet de gérer cela très simplement.

ouvrir le projet dans : **parametres-non-volatiles-settings**

le compiler et l'exécuter 2 fois pour voir le résultat.

Avez vous des questions ?

ii. Mise en œuvre de timer, QTimer

Pour qu'un programme fasse des actions en continu, comme une animation graphique par exemple, la première idée qui vient à l'esprit consiste à mettre ses actions dans une boucle sans fin. Hélas, cette manière de faire bloque le programme interdisant tous les événements utilisateur (frappe au clavier, action souris, .etc.). Il n'est alors plus possible de rien faire, même pas de l'arrêter !

Le solution à ce problème est de mettre en œuvre un *timer.* La classe **QTimer** fait ça à merveille.

ouvrir le projet dans : **utilisation-et-mise-en-oeuvre-de-timer**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

Rajouter sur l'interface (dans une QSpinbox) une valeur cible pour le compteur et modifier le programme pour que le timer s'arrête quand la valeur cible est atteinte.

À vous de travailler !

Reprendre le programme du feux rouge, demandé au paragraphe V.B.vi, page 12, pour que le feux s'anime automatiquement au moyen d'un timer. (Durées: 5 secondes pour rouge et vert, 1 seconde pour orange).

Marche à suivre: utiliser un timer cadencé à 1 seconde qui modifie une variable **`etat`**, (0=vert, 1=orange et 2=rouge), suivant la règle: **`etat=(etat+1)%3`**.

iii. Génération d'impression ou de document pdf

Le titre est suffisamment explicite :

Un programme qui imprime des états directement sur l'imprimante (avec gestion des paramètres d'impression bien entendu) ou un programme qui génère des états au format *acrobat* PDF.

ouvrir le projet dans : **ecriture-formattage-de-fichier-pdf**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

À vous de travailler !

Créez une application graphique qui permet de saisir toutes les informations relatives à un étudiant de la promotion (numéro, nom, prénom, date de naissance, adresse, photo), puis crée un fichier pdf représentant sa fiche signalétique avec photo.

Le fichier portera *le_numéro*.pdf comme nom.

ouvrir le projet dans : **impression-d-etats-sur-imprimante**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

iv. Applications *multithread*, Qthread

En informatique, le *multitâche* est une méthode où plusieurs tâches, aussi appelées processus, partagent des ressources de traitement communes comme une unité centrale. Avec un système d'exploitation multitâche, comme Linux ou Windows, vous pouvez lancer plusieurs applications simultanément. Le multitâche fait référence à la capacité d'un système d'exploitation à passer rapidement d'une tâche informatique à l'autre pour donner l'impression que les différentes applications sont en train d'exécuter plusieurs actions simultanément.

Le *multithread* étend l'idée du multitâche aux applications, de sorte que vous pouvez sous-diviser des opérations spécifiques au sein d'une même application en *threads* individuels. Chaque *thread* peut fonctionner en parallèle. Le système d'exploitation divise le temps de traitement non seulement entre différentes applications, mais aussi entre chaque thread dans une même application.

Un exemple d'application peut être divisé en quatre threads : un thread d'interface utilisateur, un thread d'acquisition de données, une communication en réseau et un thread d'enregistrement. Vous pouvez établir des priorités entre eux afin qu'ils fonctionnent séparément. Ainsi, dans des applications *multithread*, plusieurs tâches peuvent s'effectuer en parallèle avec d'autres applications qui fonctionnent sur le système. (**source: http://www.ni.com/white-paper/6424/fr/#toc3**),

ouvrir le projet dans : **utilisation-des threads**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

Dans une application GUI.

ouvrir le projet dans : **utilisation-threads-graphique**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

v. Applications en réseau *broadcast et capture UDP*

Le ***User Datagram Protocol*** (**UDP**) est un des principaux protocoles de télécommunication utilisés par Internet. Il fait partie de la couche transport du modèle OSI, il appartient à la couche 4, comme TCP. Il est détaillé dans la RFC 768

Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port. Contrairement au protocole TCP, il fonctionne sans négociation : il n'existe pas de procédure de connexion préalable à l'envoi des données (le *handshaking*). Donc UDP ne garantit pas la bonne livraison des datagrammes à destination, ni leur ordre d'arrivée. Il est également possible que des datagrammes soient reçus en plusieurs exemplaires.

L'intégrité des données est assurée par une somme de contrôle sur l'en-tête. L'utilisation de cette somme est cependant facultative en IPv4 mais obligatoire avec IPv6. Si un hôte n'a pas calculé la somme de contrôle d'un datagramme émis, la valeur de celle-ci est fixée à zéro. La somme de contrôle inclut les adresses IP source et destination.

La nature de UDP le rend utile pour transmettre rapidement de petites quantités de données, depuis un serveur vers de nombreux clients ou bien dans des cas où la perte d'un datagramme est moins gênante que l'attente de sa retransmission. Le DNS, la voix sur IP ou les jeux en ligne sont des utilisateurs typiques de ce protocole.

(source : https://fr.wikipedia.org/wiki/User_Datagram_Protocol)

Un serveur qui *broadcast* les échantillons d'une sinusoïde sur le port 10001 de la machine locale (*localhost*)

ouvrir le projet dans : **broadcastUDPdatagram**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

Un client qui capture les *datagram* ci-dessus, sur la machine locale également, et les affiche sur la console.

ouvrir le projet dans : **captureUDPdatagram**

le compiler et l'exécuter pour voir le résultat. ***Avez vous des questions ?***

À vous de travailler

Écrire un serveur qui envoie en UDP, sous la forme d'un *datagram* contenant la date et l'heure de sa machine, toute les secondes.

Écrire le client UDP qui affiche la date et heure reçues sur une interface de votre choix.

L'adresse du client, et le port sont sauvegardés par le serveur en utilisant la classe QSettings - *cf.* paragraphe VI.A.i, page 14.

vi. Application client/serveur TCP

Transmission Control Protocol, abrégé TCP, est un protocole de transport fiable, en mode connecté, documenté dans la RFC 7931 de l'IETF.

Dans le modèle Internet, aussi appelé modèle TCP/IP, TCP est situé au-dessus de IP. Dans le modèle OSI, il correspond à la couche transport, intermédiaire de la couche réseau et de la couche session. Les applications transmettent des flux de données sur une connexion réseau. TCP découpe le flux d'octets en *segments* dont la taille dépend de la MTU du réseau sous-jacent (couche liaison de données).

Un serveur TCP que vous pourrez tester avec la commande *telnet*:

```
telnet localhost 10000
```

tapez X, puis Y, puis autre chose, et enfin Q

ouvrir le projet dans : **serveur-TCP**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

À vous de travailler !

En s'aidant de l'exemple ci-dessus, écrire un serveur TCP qui attend que lui client lui envoie le caractère 'd' pour lui envoyer la date, ou 'h' pour lui envoyer l'heure, ou 'q' pour se déconnecter. Tester avec telnet.

VII. Application utilisant des bibliothèques externes

L'utilisation de bibliothèques externes n'est pas tout à fait dans le thème de ce document qui était « le framework Qt », mais j'ai tenu à en parler car l'utilisation conjointe de Qt et d'autres bibliothèques ouvre des horizons infinis. A titre personnel, il m'est arrivé de mettre en œuvre dans une même application Qt plus d'une dizaine de bibliothèques (libusb, libalsa, libespeak, opencv,

libdmtx, *.etc*.).

Le problème à ce stade n'est plus d'avoir des idées et de se poser la question « est-ce que c'est faisable ? », mais « comment trouver le temps pour le faire ? » ; 24 heures par jour, c'est trop court !

Qt vous permettra de faire tout ce que vous pouvez imaginer, mais également ce que vous n'avez pas encore imaginé.

i. Utilisation de la bibliothèque libsensors

Pour commencer, j'ai choisi une petite bibliothèque *amusante* qui permet d'accéder aux capteurs matériels de votre ordinateur: les températures, tensions, vitesses de ventilateurs, *.etc*.

Installation sous Linux

```
sudo apt-get install libsensors4-dev
```

Fichiers installés :

```
/usr/include/sensors/error.h
/usr/include/sensors/sensors.h
/usr/lib/x86_64-linux-gnu/libsensors.a
/usr/lib/x86_64-linux-gnu/libsensors.so
/usr/share/doc/libsensors4-dev/changelog.Debian.gz
/usr/share/doc/libsensors4-dev/copyright
/usr/share/doc/libsensors4-dev/libsensors-API.txt.gz
/usr/share/man/man3/libsensors.3.gz
```

Pour pouvoir lier (*linker*) avec notre application il faut juste ajouter dans le fichier du projet (.pro) les lignes suivantes permettant de spécifier le chemin d'inclusion des headers (.h) et l'ajout de la bibliothèque *libsensors* à l'exécutable. Ceci permettra d'ajouter les options à la ligne de commande du compilateur : `-I/usr/include/sensors, (-L/usr/lib/x86_64-linux-gnu) et -lsensors`

Lignes à rajouter dans le .pro :

```
##   UTILISATION DE LA BIBLIOTHEQUE libsensors
INCLUDEPATH  += /usr/include/sensors/
LIBS   += -lsensors
```

Il ne reste plus qu'à consulter la documentation de la bibliothèque, (commande : `man libsensors`) pour y trouver la liste des fonctions, leurs actions, et leurs prototypes, *.etc.*

Un exemple permettant d'afficher les valeurs disponibles :

ouvrir le projet dans : **exemple-libsensors-console**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

Le même avec une interface qui se construit en fonction des senseurs disponibles. Un peu plus difficile au niveau Qt ! Dans un premier temps on construit l'interface (dans le constructeur) puis un **timer** met à jour les valeurs.

ouvrir le projet dans : **exemple-libsensors-GUI**

le compiler et l'exécuter pour voir le résultat. Avez vous des questions ?

ii. Représentation graphique de données avec QWT

La bibliothèque QWT permet de mettre en œuvre des composants visuels de haut niveau tels que : des graphiques avec courbes, des boussoles, des compteurs, des horizons artificiels, *.etc.*

Vous pouvez trouver qwt à : http://qwt.sourceforge.net/

Comme précédemment, il faut tout d'abord installer la bibliothèque. Le problème cette fois-ci est qu'elle n'est pas disponible dans les dépôts de Linux. Il faut donc repartir des codes sources et la recompiler localement. A l'heure où j'écris ce document, la dernière version est la 6.1.3 du 12/06/2016.

Compilation et installation :

1. Télécharger les codes sources à : https://sourceforge.net/projects/qwt/files/latest/download?source=files
2. Décompresser l'archive, dans un répertoire de votre choix (moi j'utilise mon *home* pour ce genre de choses)
3. ouvrir le fichier de projet (`qwt.pro`) avec QtCreator
4. construire dans un répertoire de votre choix (**`build`** sous le répertoire de qwt par exemple). Vous pouvez aller boire un café...
5. Se rendre dans le répertoire ci-dessus avec un terminal puis entrez la commande :

   `sudo make install`.
6. La bibliothèque sera installée par défaut sous : `/usr/local/qwt-6.1.3`

Comme précédemment il faudra rajouter cette bibliothèque dans le fichier de projet :

```
#   QWT library - must be installed
INCLUDEPATH += /usr/local/qwt-6.1.3/include
LIBS += -L/usr/local/qwt-6.1.3/lib  -lqwt
```

Pour chercher les *headers* à **`/usr/local/qwt-6.1.3/include`** et pour lier la bibliothèque ***libqwt.so*** qui se trouve dans le répertoire **`/usr/local/qwt-6.1.3/lib`**

Voilà, il ne reste plus qu'à !

ouvrir le projet dans : **qwt-plot-des-belles-courbes**

le compiler et l'exécuter pour voir le résultat. **Avez vous des questions ?**

Vous trouverez sur mon site d'autres exemples :

http://grimaldi.univ-tln.fr/category/files/qwt-attitude-horizon-artificiel.zip

http://grimaldi.univ-tln.fr/category/files/qwt-compas-une-boussole.zip

http://grimaldi.univ-tln.fr/category/files/qwt-plot-figures-de-Lissajou.zip

http://grimaldi.univ-tln.fr/category/files/qwt-speedo-un-compteur-de-vitesse.zip

À vous de travailler

En réutilisant les codes du client et du serveur *UDP* précédent - cf. paragraphe VI.A.v, page 16 - modifier le client pour qu'il trace la sinusoïde envoyée en temps réel sur un graphique qwt.

iii. Transformée de Fourier avec libfftw

LibFFTW est une bibliothèque de sous-programmes C permettant de calculer la transformée de Fourier discrète (DFT) en une ou plusieurs dimensions, avec une taille d'entrée arbitraire, pour des données réelles et complexes (ainsi que de données paires ou impaires, c'est-à-dire les transformées cosinus / sinus discret, DCT / DST).

Installation : elle est disponible dans les dépôts de L**inux** en version 3 :

```
sudo apt-get install libfftw3-dev
```

Fichiers installés :

```
/usr/bin/fftw-wisdom
/usr/bin/fftw-wisdom-to-conf
/usr/bin/fftwf-wisdom
/usr/bin/fftwl-wisdom
/usr/include/fftw3.f
/usr/include/fftw3.f03
/usr/include/fftw3.h
/usr/lib/libfftw3.a
/usr/lib/libfftw3.so
/usr/lib/libfftw3_threads.a
/usr/lib/libfftw3_threads.so
/usr/lib/libfftw3f.a
/usr/lib/libfftw3f.so
/usr/lib/libfftw3f_threads.a
/usr/lib/libfftw3f_threads.so
/usr/lib/libfftw3l.a
/usr/lib/libfftw3l.so
/usr/lib/libfftw3l_threads.a
/usr/lib/libfftw3l_threads.so
/usr/lib/pkgconfig/fftw3.pc
/usr/lib/pkgconfig/fftw3f.pc
/usr/lib/pkgconfig/fftw3l.pc
/usr/share/doc/libfftw3-dev/changelog.Debian.gz
/usr/share/doc/libfftw3-dev/copyright
/usr/share/doc/libfftw3-dev/examples/Makefile.am
/usr/share/doc/libfftw3-dev/examples/Makefile.in
/usr/share/doc/libfftw3-dev/examples/README
/usr/share/doc/libfftw3-dev/examples/bench.c
/usr/share/doc/libfftw3-dev/examples/check.pl
/usr/share/doc/libfftw3-dev/examples/fftw-bench.c
```

```
/usr/share/doc/libfftw3-dev/examples/hook.c
/usr/share/man/man1/fftw-wisdom-to-conf.1.gz
/usr/share/man/man1/fftw-wisdom.1.gz
/usr/share/man/man1/fftwf-wisdom.1.gz
/usr/share/man/man1/fftwl-wisdom.1.gz
```

ouvrir le projet dans : **exemple-utilisant-libfftw**

le compiler et l'exécuter pour voir le résultat.
Avez vous des questions ?

Le programme d'exemple précédent permet à l'utilisateur de choisir un signal temporel (sinus, carré, pulse ou bruit blanc) puis d'en calculer la transformée de Fourier.
Une représentation graphique utilisant la bibliothèque QWT du signal et du spectre est donnée en temps réel. (attention pensez à vérifier que la version de qwt dans le fichier de projet corresponde bien à celle installée sur votre machine)

À vous de travailler

En réutilisant les codes du client et du serveur *UDP* précédent- cf. paragraphe VI.A.v, page 16 - modifier le client pour qu'il trace la sinusoïde envoyée et son spectre de Fourier, en temps réel sur un graphique qwt

iv. Traitement d'image avec Opencv

Si vous voulez faire du traitement d'images allant de la simple acquisition, jusqu'aux derniers algorithmes du moment, je ne saurais vous conseiller OpenCV.

A ce stade, je me contenterai juste de citer cette bibliothèque.

- Téléchargement : http://opencv.org/
- Installation sous Linux : http://docs.opencv.org/3.1.0/d7/d9f/tutorial_linux_install.html
- Documentation enligne : http://docs.opencv.org/3.1.0/index.html

Ensuite, le web foisonne d'exemples, de forums, de tutoriaux, .etc. - pour utilisateur averti !

À vous de travailler

Écrire un programme qui fait l'acquisition des images de votre webcam est les affiche sur l'interface. (en 15 minutes, une fois OpenCV installée)

À vous de travailler

Écrire un programme qui fait l'acquisition des images et les broadcast en UDP sur le réseau (en 30 minutes, ce coup ci . . .)

VIII. Conclusion

Comme je vous l'écrivais précédemment, l'environnement Qt peut vous permettre d'écrire tous les programmes que vous pouvez imaginer, mais aussi ceux que vous ne pouvez pas imaginer, et tous ceux que vous imaginerez dans les années à venir.

Je n'ai pas parlé de la *cross-compilation* qui vous permet de porter vos applications Qt sur toutes sortes de plateformes linux (ou autre) : raspberry PI, beaglebord, pandabord, *.etc*, *.etc*, *.etc*, ou encore d'écrire des applications mobiles sur **android** ou **IOS**.

Enfin, et c'est un détail que j'avais complètement omis tant il est peu important, Le framework Qt est également implémenté sous Windows et sur MacOS. Tout ce qui précède (enfin presque) peut être fait sur le magnifique système d'exploitation de Microsoft (je vous souhaite bien du courage).