Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs
Systèmes Numériques option
Informatique et Réseaux

Tutoriel IHM sous Qt

Projet Alarme Domestique

2ère Séquence : Clavier et Détecteurs

Date : Octobre 2022
Version : 4.2
Référence : S2 - CentraleAlarme – Clavier et Detecteurs

1. Objectif

- *Tutoriel pour la gestion de l'IHM sous **Qt***
- *Utilisation de la documentation en ligne de **Qt***
- *Prise en main de l'environnement de développement*
- *Réalisation d'un formulaire **Qwidget** composé de **QPushButton** et de **Qcheckbox***
- *Mise en place de **Layout***
- *Utilisation de boîte de Message **QMessageBox***
- *Réalisation d'une boîte de dialogue **QDialog***
- *Notion de signal et slot issu de la classe **QObject***
- *utilisation d'un timer **QTimer***

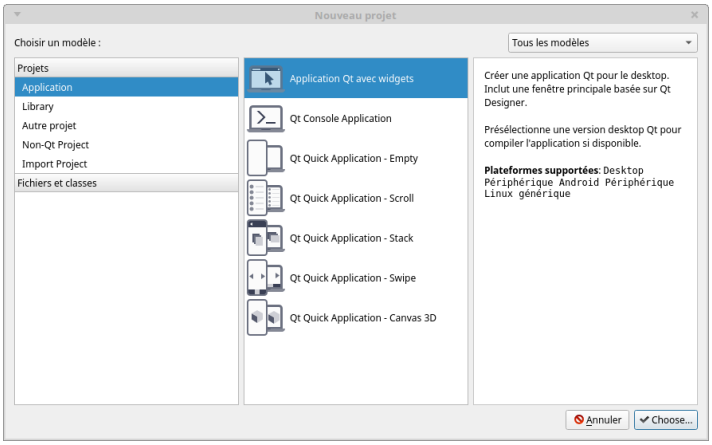
2. Conditions de réalisation

- *Ce fichier contient des liens hypertextes.*
- *Ressources utilisées :*

Un PC sous Linux	*Qt-creator*

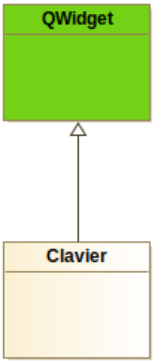
3. Création du projet

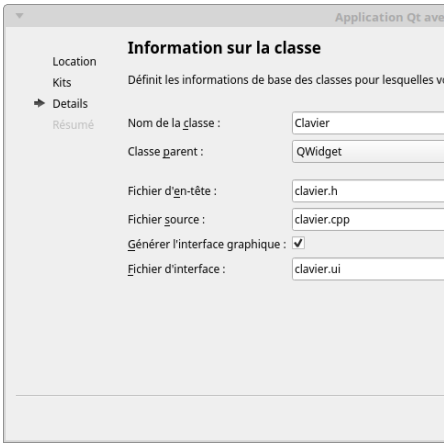
*Après avoir lancé **Qt creator**, choisir dans le menu **fichier** l'option **Nouveau fichier ou projet**. Choisir ensuite une **Application Qt avec widgets** comme le montre la figure.*



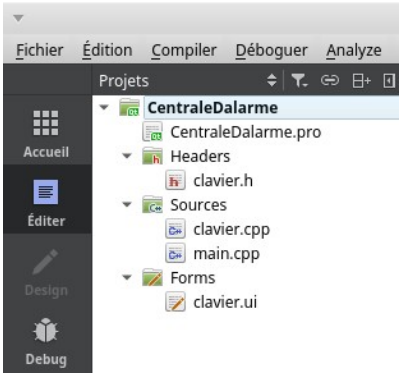
Ce projet se nomme **CentraleDalarme**, il est placé sur le disque de travail dans votre dossier **ProjetQt** et utilise le suivi de version Git

Dans un premier temps, réaliser la classe **Clavier** dont le parent sera de type **QWidget**. (relation d'héritage).





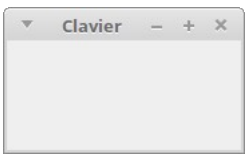
L'EDI QtCreator fabrique les fichiers suivant :



CentraleDalarme.pro	Contient la structure du projet
clavier.h	Déclaration de la classe Clavier
clavier.ui	Contient l'interface utilisateur
clavier.cpp	Code de la classe Clavier
main.cpp	Programme principal

Vous n'avez pas à intervenir dans le fichier **main.cpp.** Son rôle est de déclarer une instance de la classe **Clavier** et d'afficher l'interface utilisateur et d'exécuter l'application.

L'icône compile et lance le programme.

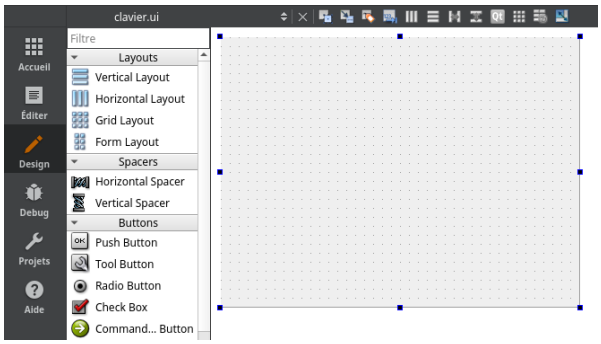



```
#include "clavier.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Clavier w;
    w.show();

    return a.exec();
}
```

Un double clic sur le fichier permet de l'éditer. Le fichier **CentraleDalarme.pro** est réalisé également automatiquement. Pour l'instant, il n'est pas à modifier, il contient les éléments nécessaires à la compilation du projet.

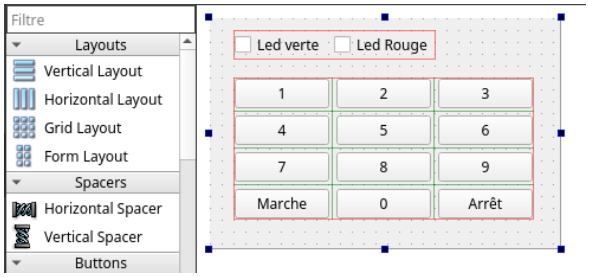
Le fichier **clavier.ui** lance la partie design et permet de fabriquer l'interface graphique.

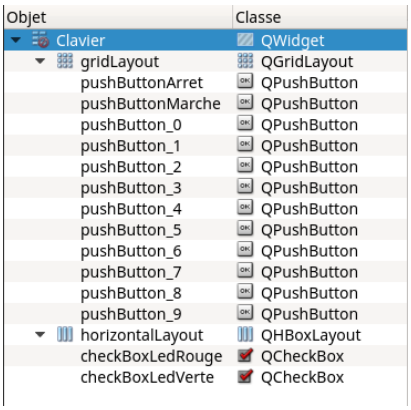
La première partie de l'écran contient les Widgets à déposer sur la grille de la partie centrale. Les propriétés de chaque élément se trouvent sur la partie droite.



4. Réalisation de l'interface utilisateur (l'aspect visuel)

L'interface est composée de 12 ***pushButton*** *et de 2* ***checkBox*** *comme le montre la figure :*

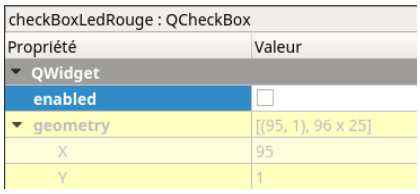




Chaque widget fait l'objet d'un glissé vers la grille de construction de l'IHM.

Ils sont renommés dans la partie propriété de l'écran. Les noms sont choisis de manière cohérente par rapport à l'application. Le fait de garder le type d'objet en préfixe permet de savoir sur quel objet le programmeur travaille.

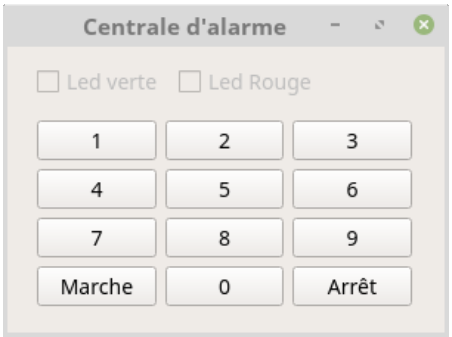
Les deux cases à cocher sont regroupées dans un ***Horizontal Layout*** et les boutons du clavier dans un ***Grid Layout***.



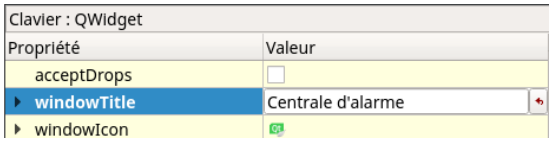
Toujours dans la zone de propriété pour les deux **checkBox**, retirez la possibilité que l'utilisateur puisse cocher la case (ici, nous souhaitons juste visualiser l'état de la centrale d'alarme).

Propriété : ***enabled***

Après exécution, vous obtenez la boîte de dialogue suivante :



Pour modifier le titre de la fenêtre, après avoir sélectionné la grille représentant l'interface utilisateur, dans les propriétés, il est possible de changer le titre et toutes les autres propriétés.



Ces propriétés seront également modifiables par programmation.

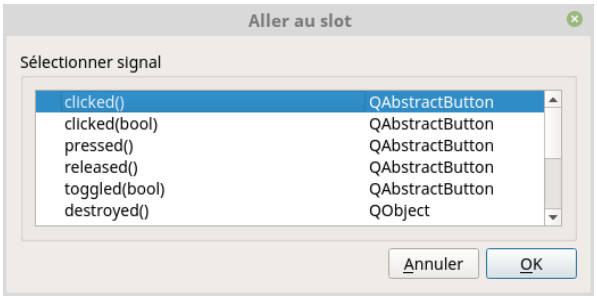
5. Association automatique d'un signal issu d'un bouton à un slot

Pour qu'un bouton réagisse, il est nécessaire d'associer un **signal** issu de ce bouton à un **slot**. Un clic droit de la souris sur le bouton « **Marche** » de l'interface fait apparaître un menu surgissant, en sélectionnant l'option **aller au slot...** la boîte de dialogue suivante s'affiche :

Un signal : Message envoyé par un widget lorsqu'un évènement se produit.

Un slot : Fonction membre appelée lorsqu'un évènement s'est produit.

UML : Message asynchrone ⟶



Pour notre application, trois signaux vont nous intéresser pour le bouton « Marche » :

clicked()	Lorsque le bouton est simplement actionné
pressed()	Lorsque le bouton est enfoncé
released()	Lorsque le bouton est relâché

Après la sélection du signal **clicked()** et validation avec le bouton « OK », la méthode suivante correspondant au slot est créée dans la classe clavier. Reste maintenant à programmer le contenu de la méthode pour réagir à l'appui sur le bouton « **Marche** ».

Dans le fichier clavier.cpp

```
void Clavier::on_pushButtonMarche_clicked()
{

}
```

Dans la déclaration de la classe clavier (clavier.h)

```
private slots:
    void on_pushButtonMarche_clicked();
```

Par exemple : Après avoir inclus le fichier <QMessageBox>, ajouter dans le corps de la méthode les éléments suivants :

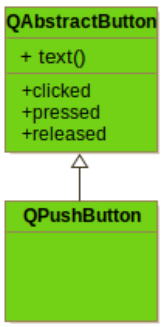
```
QMessageBox messageMarche;
messageMarche.setText("j'ai appuyé sur Marche");
messageMarche.exec();
```

Compilez et exécutez le programme, après appui sur le bouton « **Marche** », la boîte de message apparaît.

Il est possible de récupérer le texte d'un bouton pour l'utiliser dans la méthode.. Remplacez le code précédent par les instructions suivantes :

```
QString texteBouton = ui->pushButtonMarche->text();
QMessageBox messageMarche;
messageMarche.setText("j'ai appuyé sur la touche " + texteBouton);
messageMarche.exec();
```

ui est un pointeur sur l'interface utilisateur, il donne accès aux différents widgets présents sur la vue. Ici, ***pushButtonMarche*** est un pointeur sur ***QAbstractButton*** qui offre toutes sortes de méthode pour agir sur notre bouton notamment dans le cas présent la récupération du texte présent sur le bouton. On y retrouve également les différents signaux qui vont être utilisés pour la suite de l'application.



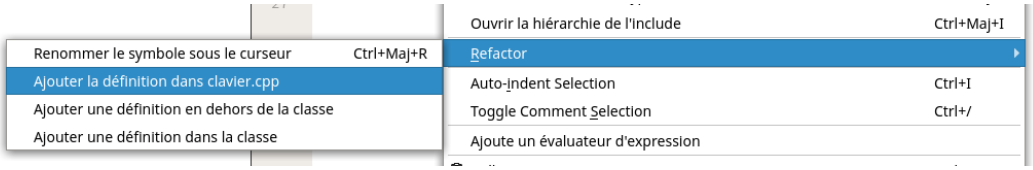
6. Association manuelle d'un *signal* à un *slot*

Dans le cas précédent, l'association du signal et du slot a été réalisée de manière complètement transparente. Aucune intervention du programmeur n'est nécessaire si ce n'est le choix du signal.

Dans le cas présent, pour les touches 0 à 9, le traitement est identique. On peut créer de manière automatique l'association ***signal*** – ***slot*** et coder 1à fois la même méthode ou créer un ***slot*** unique void Clavier::TraiterChiffre() qui exécutera le même code pour chaque chiffre. Pour cela, dans la déclaration de la classe, ajoutez la déclaration du slot :

```
private slots:
   void on_pushButtonMarche_clicked();
   void TraiterChiffre();
```

puis avec un clic droit sur la méthode **TraiterChiffre()** fait apparaître les menus surgissants suivant :



Ils permettent d'ajouter sans erreur l'implémentation de la méthode dans le fichier .cpp. Complétez par le code C++ correspondant au traitement voulu.

```
void Clavier::TraiterChiffre()
{
       QPushButton *pbouton = static_cast<QPushButton *>(sender());
       QString texteBouton = pbouton->text();
       QMessageBox messageChiffre;
       messageChiffre.setText("j'ai appuyé sur la touche " + texteBouton);
       messageChiffre.exec();
}
```

La difficulté maintenant est de savoir qu'elle touche a envoyé le signal ***clicked()***. Ce problème est résolu par la première ligne. La méthode ***sender()*** dont la classe **Clavier** a hérité de la classe **QObject** retourne un pointeur sur l'émetteur d'un signal.

Reste maintenant à connecter, les signaux **clicked()** de chaque touche chiffre au slot void Clavier::TraiterChiffre() dans le constructeur de la classe **Clavier**.

```
Clavier::Clavier(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::Clavier)
{
   ui->setupUi(this);
   connect(ui->pushButton_0,&QPushButton::clicked,this,&Clavier::TraiterChiffre);
   connect(ui->pushButton_1,&QPushButton::clicked,this,&Clavier::TraiterChiffre);
   // poursuivre avec les autres touches
}
```

Objet récepteur

Nom du slot

Objet émetteur

Nom du signal

La méthode **connect()** héritée de la classe **QObject** réalise ce mécanisme.

Voici le résultat obtenu après compilation et exécution :

7. Mise à jour des cases à cocher

Comme nous avons pu le constater, le pointeur **ui** donne accès aux éléments de l'interface graphique. Ainsi :

ui->checkBoxLedRouge->checkState()	Donne l'état de la LED Rouge
ui->checkBoxLedRouge->setCheckState(...);	Fixe l'état de la LED Rouge

Les valeurs possibles pour l'état d'une case à cocher sont : Qt::Checked ou Qt::Unchecked, deux constantes définis par Qt.

À vous de jouer

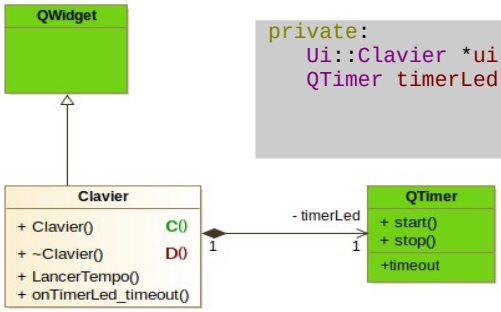
En vous inspirant des explications du paragraphe 5, modifiez le code du slot ***on_pushButtonMarche_Clicked*** pour que l'appui sur la touche **Marche** coche la case à cocher représentant la LED Rouge.

Réalisez les opérations nécessaires pour que la touche **Arrêt** la décoche.

Dans un deuxième temps, on souhaite faire clignoter automatiquement une LED. Le framework ***Qt*** dispose d'une classe ***QTimer*** permettant de répondre à ce besoin.

L'utilisation de cette classe est décrite en suivant le lien : https://doc.qt.io/qt-5/qtimer.html

Dans la section privée de déclaration de la classe **Clavier**, ajoutez un objet **QTimer** nommé **timerLed** afin de coder la relation entre la classe **Clavier** et la classe **Qtimer**.




```
private:
   Ui::Clavier *ui;
   QTimer timerLed;
```

Ajouter également la ligne :

```
#include <QTimer>
```

Dans la documentation de **Qt**, les signaux sont indiqués comme ceux-ci :

Signals ¶

void timeout()

Dans le constructeur de la classe Clavier, le signal **timeout()** (temps écoulé) issu de l'objet **QTimer** doit être connecté à un **slot** de traitement. Celui-ci se nommera **onTimerLed_timeout()** par convention. Il est préférable de commencer par ajouter ce slot à la déclaration de la classe puis d'ajouter son implémentation dans le fichier **clavier.cpp** avec **Refractor** (voir paragraphe 6).

La connexion dans le constructeur, comme précédemment se fait de la manière suivante :

```
connect(&timerLed,&QTimer::timeout,this,&Clavier::onTimerLed_timeout);
```

Le premier paramètre est **&timerLed** car la fonction **connect** demande l'adresse de l'objet émetteur, comme ici **timerLed** n'est pas un pointeur, il est nécessaire d'utiliser l'opérateur **&** pour en obtenir son adresse.

Modifiez dans les slots correspondants aux actions des boutons **Marche** et **Arrêt** les commandes pour lancer et arrêter le **timerLed** (voir documentation QTimer) **:**

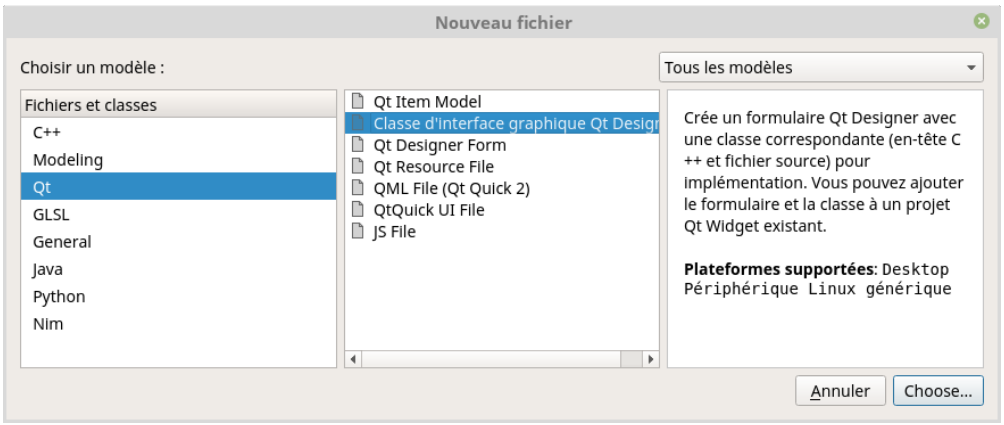
`void start(int msec)`	Pour lancer le timer : timerLed.start(500);
`void stop()`	Pour arrêter le timer : timerLed.stop();

Le code du slot appelé à l'issue de 500 ms est donné ici :

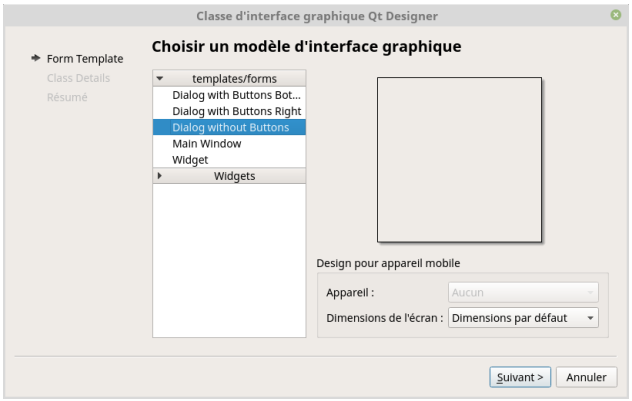
```
void Clavier::onTimerLed_timeout()
{
   if(ui->checkBoxLedRouge->checkState() == Qt::Checked)
       ui->checkBoxLedRouge->setCheckState(Qt::Unchecked);
   else
       ui->checkBoxLedRouge->setCheckState(Qt::Checked);
}
```

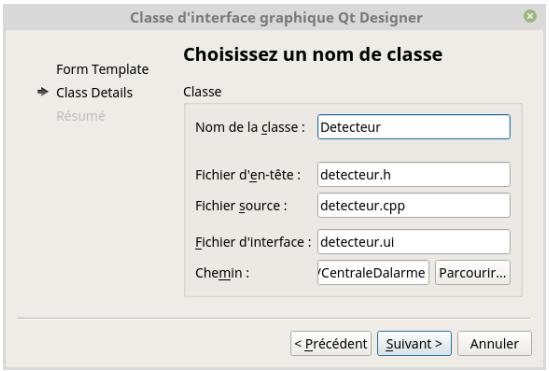
8. Affichage d'une nouvelle boîte de dialogue (le détecteur)

Dans la fenêtre Projet, à partir du bouton droit de la souris sur la racine dans l'explorateur, sélectionnez l'option **Ajouter nouveau...** pour faire apparaître la fenêtre suivante et sélectionner une **Classe d'interface graphique Qt Designer**.

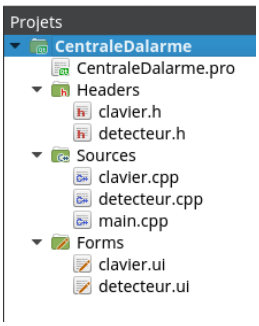


Puis après avoir appuyé sur le bouton **Choose…** sélectionnez **Dialog without Buttons** et Suivant.

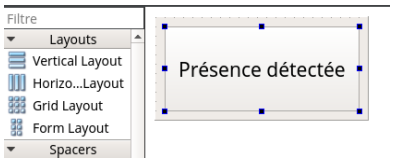




Nommez cette nouvelle classe **Detecteur**. L'appui sur Suivant complète le projet avec les fichiers :



Modifiez le fichier detecteur.ui pour obtenir la figure suivante :



Le nouveau widget contient un bouton nommé **pushbuttonIntrus**.

La police peut être changée dans les propriétés du bouton.

À vous de jouer

De quelle classe hérite la classe Detecteur ?

Dans la classe **Clavier** (dans le cadre du tutoriel uniquement), ajoutez un pointeur sur la classe **Detecteur** nommé **leDetecteur**. Réalisez les traitements nécessaires pour que la touche **2** fasse apparaître la nouvelle boîte et la touche **3** la fasse disparaître. (vous devrez peut-être supprimer certains **connect** du constructeur de la classe clavier).

Pour faire apparaître le détecteur on propose 2 solutions, expliquez la différence.

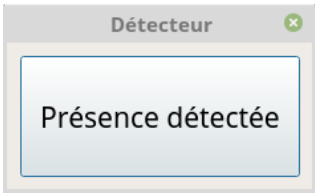
```
void Clavier::on_pushButton_2_clicked()
{
    leDetecteur = new Detecteur;
    leDetecteur->exec();
}
```

```
void Clavier::on_pushButton_2_clicked()
{
    leDetecteur = new Detecteur;
    leDetecteur->show();
}
```

Voir l'aide https://qt.developpez.com/faq/?page=modules-qtgui-affichage-fenetres

Dans le constructeur de la classe Detecteur, changez le titre de la boîte avec la méthode **setWindowsTitle** en indiquant qu'il s'agit d'un détecteur :

Pour ceux qui n'ont pas trouvé comment cacher le Détecteur, on propose le code :




```
void Clavier::on_pushButton_3_clicked()
{
    leDetecteur->hide();
    delete leDetecteur;
}
```

À vous de jouer

Que se passe-t-il si l'on appuie d'abord sur la touche 3 ?

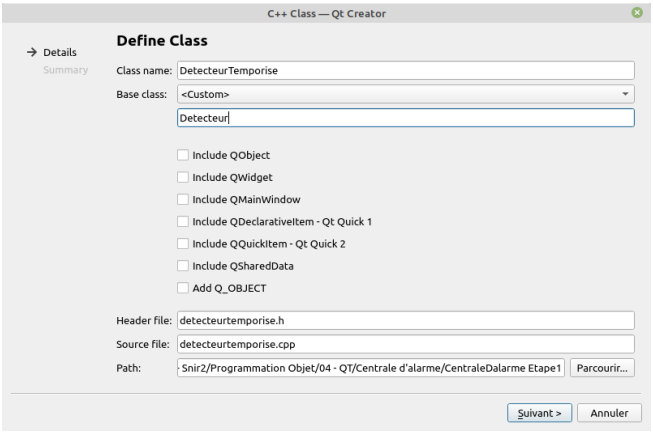
Pour pallier à ce problème on propose de tester si leDetecteur est différent de nullptr. Dans ce cas, il est nécessaire d'initialiser le pointeur leDetecteur à la valeur nullptr dans le constructeur. Lors de la création, il est bon également de tester si la valeur du pointeur est nullptr avant d'instancier dynamiquement un nouveau détecteur pour éviter de consommer de la mémoire pour rien.

9. Application

À vous de jouer

a) Faites apparaître un message Intrus détecté lors d'un appui sur le bouton **Présence détectée**.

b) Créez une **nouvelle classe C++ DetecteurTemporise** qui hérite de **Detecteur**



Le constructeur de cette classe indique qu'il s'agit d'un détecteur temporisé en titre.

L'appui sur le bouton **Présence détectée** lance une temporisation de 3 secondes.

Au bout des 3 secondes, le message «Un intrus a été détecté» apparaît.

À vous de jouer

c) Gérer l'apparition et la disparition du nouveau détecteur avec les touches 4 et 5.

d) Lors de l'appel du destructeur de la classe, arrangez-vous pour arrêter le timer, afin d'éviter l'apparition du message même lorsque le détecteur temporisé est masqué.