




# TP 24 –Signaux et Slots

Vous allez réaliser une application qui présente un unique bouton poussoir. L'utilisateur a 5 secondes pour cliquer sur ce bouton. S'il le fait, alors il a gagné, sinon, il a perdu.

## Exercice 0 Vérification de la configuration de votre IDE

 **Rappel :** vous pouvez choisir d'utiliser Visual Studio OU Qt Creator. Les instructions spécifiques pour l'un ou l'autre IDE sont identifiées par les images  et  dans ce document.



## 0.1 Vérification de l'installation de l'extension Qt Pour Visual Studio

Vérifiez que vous possédez bien l'élément « Extensions / Qt VS Tools » dans la barre de menu de Visual Studio. Si ce n'est pas le cas, suivez les instructions du tutoriel vidéo sur le cours en ligne : <https://ent.uca.fr/moodle/mod/book/view.php?id=370354&chapterid=2682>



## 0.1 Vérification de la configuration de Qt Creator

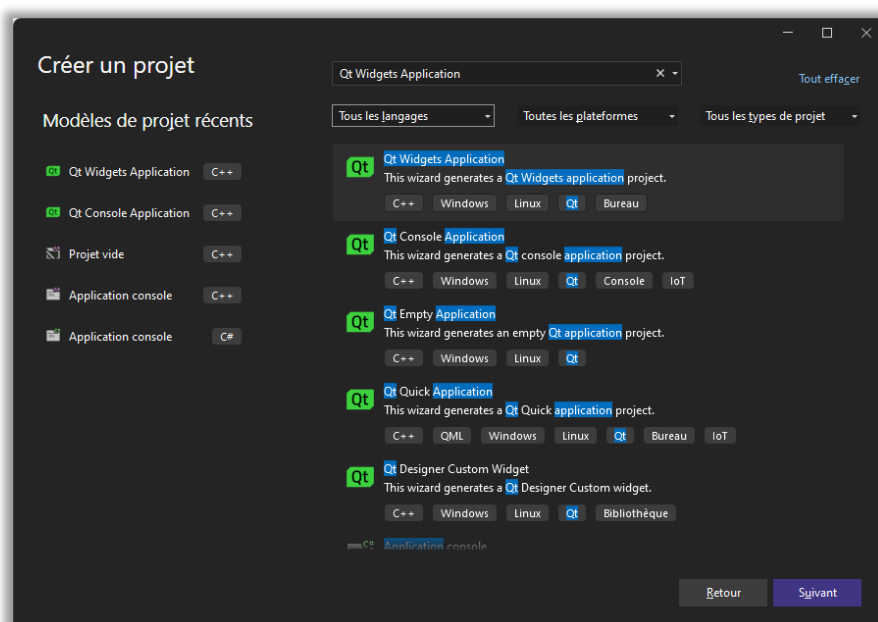
Assurez-vous d'avoir suivi les instructions du tutoriel vidéo sur le cours en ligne : <https://ent.uca.fr/moodle/mod/book/view.php?id=370354&chapterid=2683>

## Exercice 1      Création d'un projet Qt

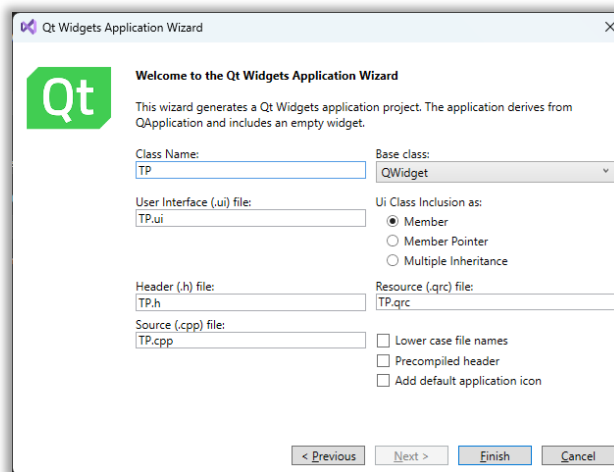


## 1.1 Travail à faire

- 📁 Créez un nouveau projet en sélectionnant le modèle **Qt Widgets Application**.
- 📁 Nommez le projet **TP**.



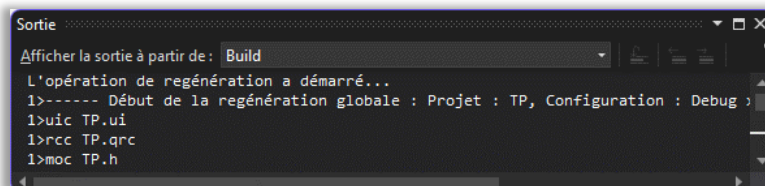
- ☞ L'assistant de projet **Qt** s'ouvre alors.
- ☞ Les deux premières étapes de l'assistant peuvent être laissées telles quelles.
- ☞ La troisième étape de l'assistant vous propose quelques configurations du code initial qui sera généré pour vous :



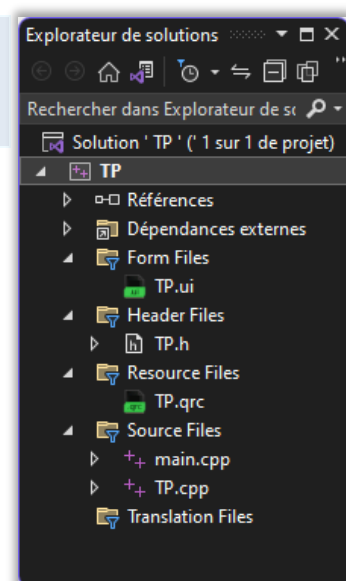
- ☞ **Class Name** est le nom de la classe principale de votre application
- ☞ **Base class** est la classe de base dont hérite la classe principale de votre application. Il s'agit par défaut d'un **QMainWindow**. Si vous laissez cette option, votre classe principale sera une fenêtre munie d'un menu, d'une barre d'outils et d'une barre de statuts.
- ☞ Choisissez ici d'hériter de la classe **QWidget**. Ainsi, la fenêtre de votre application n'aura pas de menu, de barre d'outils ni de barre de statuts.
- ☞ **.h file** est le nom du fichier de déclaration de votre classe principale
- ☞ **.cpp file** est le nom du fichier de définition de votre classe principale
- ☞ **.ui file** est le nom de déclaration de votre interface utilisateur (éditable avec **Qt Designer**)
- ☞ **.qrc file** est le nom du fichier de déclaration des ressources de votre application.

☞ L'assistant a créé pour vous une structure d'application comportant les éléments de la figure ci-contre.

- ☞ Compilez le projet
- ☞ Remarquez les trois premières étapes de la compilation :



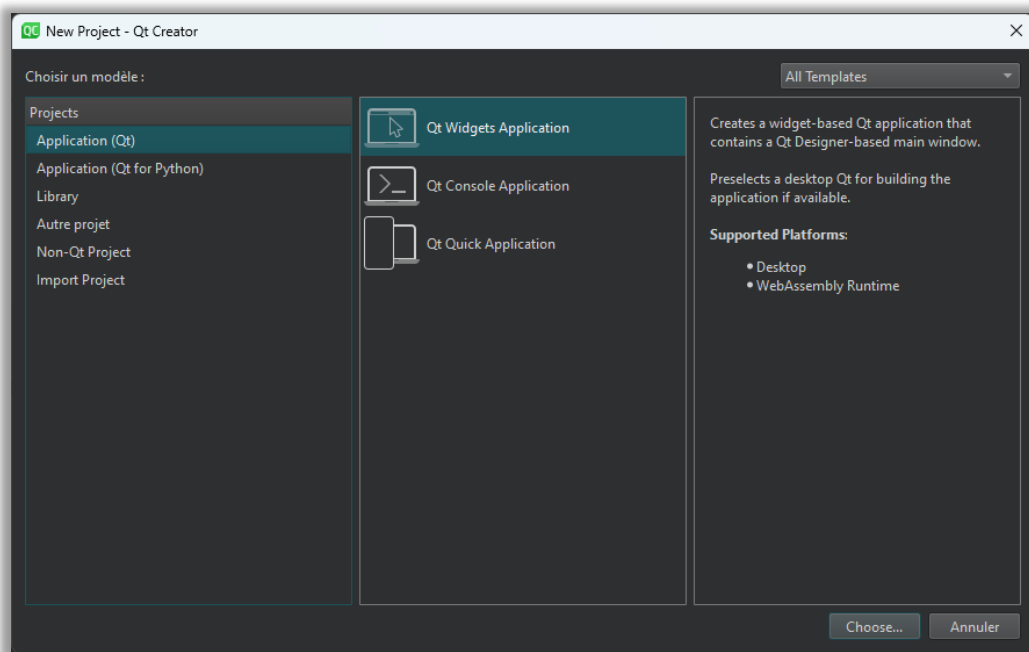
- ☞ Ouvrez le fichier **main.cpp**
- ☞ Comme vu en cours, les 4 lignes de codes initialisent Qt, créent un objet **TP** nommé **w**, l'affichent et lancent la boucle d'événement de l'application.



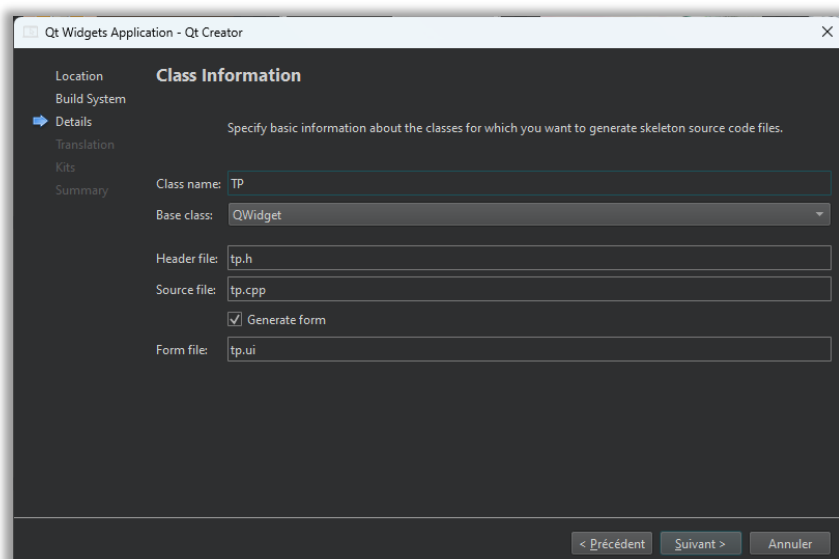
- 📄 Ouvrez les fichiers **tp.h** et **tp.cpp**
  - 🔗 Observez les données membres et le constructeur.
    - ➡ A quoi sert l'appel de la fonction **ui.setupUi(this)** ?
- 📄 Exécutez le programme
  - 🔗 Vous observez une fenêtre vide, votre point de départ.

## QC 1.1 Travail à faire

- 📄 Créez un nouveau projet en sélectionnant le modèle **Qt Widgets Application**.




- 📄 L'assistant de projet **Qt** s'ouvre alors.
  - 🔗 Nommez le projet **TP**.
  - 🔗 La seconde étape peut être ignorée.
- 📄 La troisième étape de l'assistant vous propose quelques configurations du code initial qui sera généré pour vous :



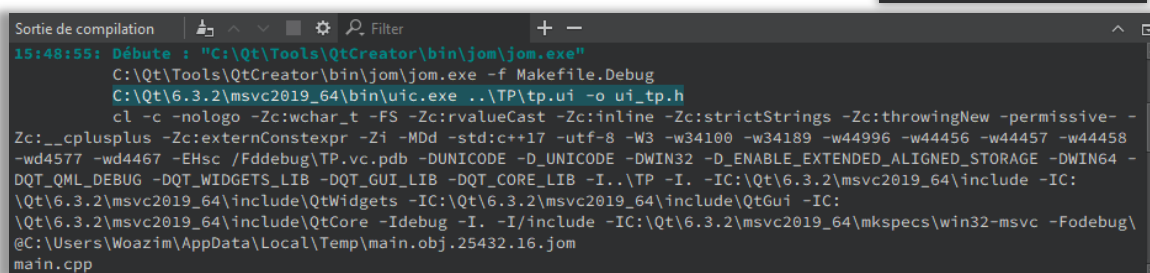
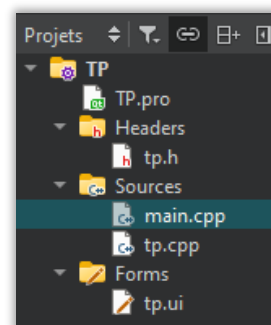
- ➡ **Class name** est le nom de la classe principale de votre application. Spécifiez TP
- ➡ **Base class** est la classe de base dont hérite la classe principale de votre application. Il s'agit par défaut d'un **QMainWindow**. Si vous laissez cette option, votre classe principale sera une fenêtre munie d'un menu, d'une barre d'outils et d'une barre de statuts.
  - ➡ Choisissez ici d'hériter de la classe **QWidget**. Ainsi, la fenêtre de votre application n'aura pas de menu, de barre d'outils ni de barre de statuts.
- ➡ **.h file** est le nom du fichier de déclaration de votre classe principale
- ➡ **.cpp file** est le nom du fichier de définition de votre classe principale
- ➡ **.ui file** est le nom de déclaration de votre interface utilisateur (éditable avec **Qt Designer**)

 Terminez l'assistant en gardant les valeurs par défaut des étapes suivantes.

 L'assistant a créé pour vous une structure d'application comportant les éléments de la figure ci-contre.

 Compilez le projet

- ➡ Remarquez les étapes de compilation spécifiques à Qt :
  - ➡ L'utilitaire **uic** sur le fichier **.ui** et l'utilitaire **moc** sur le fichier **tp.h**



 Ouvrez le fichier **main.cpp**

- 👉 Comme vu en cours, les 4 lignes de codes initialisent Qt, créent un objet **TP** nommé **w**, l'affichent et lancent la boucle d'événement de l'application.

 Ouvrez les fichiers **tp.h** et **tp.cpp**

- ➡ Observez les données membres et le constructeur.
- ➡ A quoi sert l'appel de la fonction `ui->setupUi(this)` ?

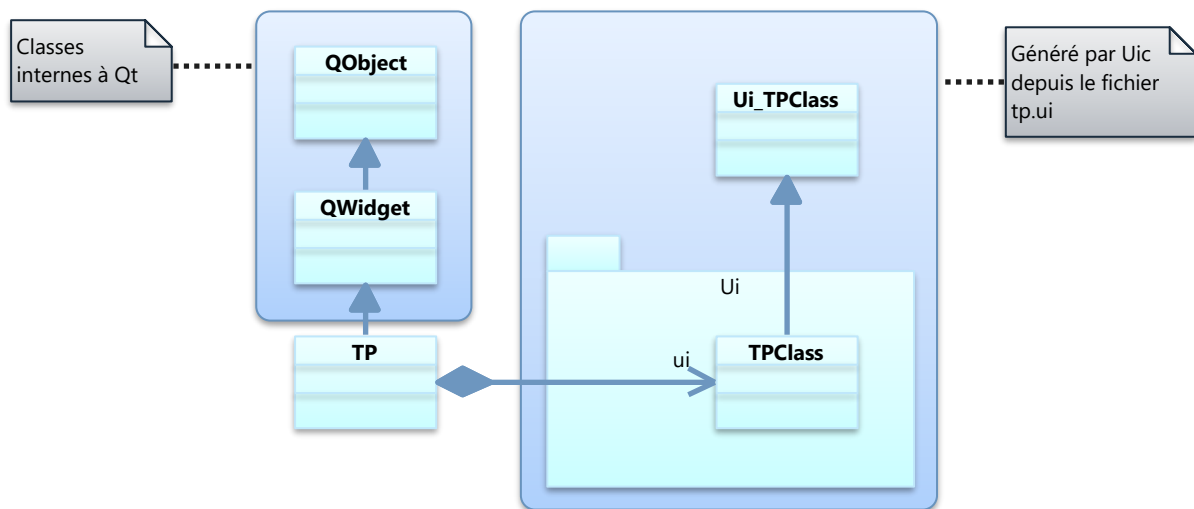
 Exécutez le programme

-  Vous observez une fenêtre vide, votre point de départ.

## 1.2 Choses à comprendre

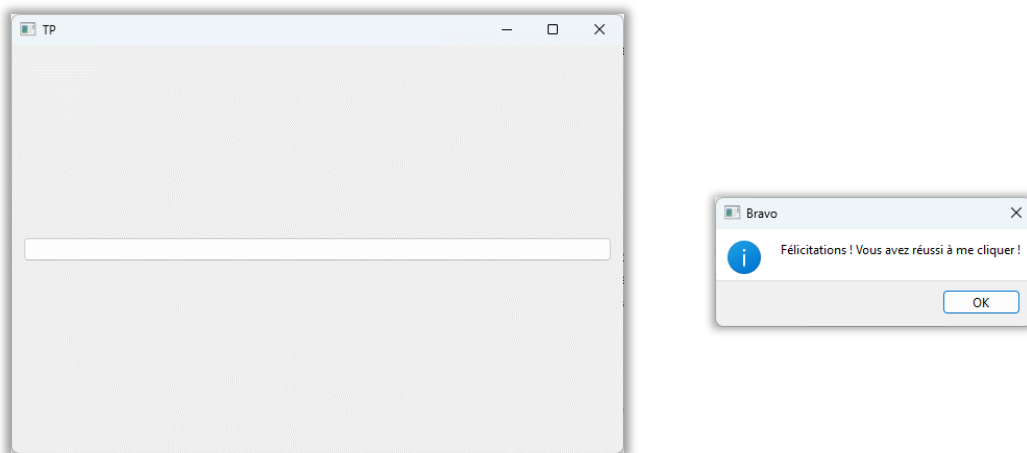
L'assistant de création de projet de Qt a créé pour vous une classe **TP** décrite dans les fichiers **tp.h** et **tp.cpp**. Cette classe hérite à travers la classe **QWidget**, de la classe principale de Qt : **QObject**. Cela lui apporte tous les mécanismes internes des **QObject**, à savoir notamment, le système composite de relation parent-enfant vu en cours. Ce mécanisme s'implémente automatiquement grâce à l'utilitaire MOC, qui produit le fichier **moc\_tp.cpp** que vous pouvez retrouver dans le répertoire « **x64\Debug\moc** » si vous utilisez Visual Studio ou dans le répertoire « **debug** » du répertoire de compilation du projet Qt Creator (il se trouve généralement un cran au-dessus du répertoire de votre projet et porte le nom « **build-TP-Desktop\*** »). Parallèlement à cela, l'interface graphique est décrite au format XML dans le fichier **tp.ui**. Ce fichier, éditable à l'aide d'un éditeur de texte, ou à l'aide de l'éditeur graphique Qt Designer, est fourni à l'utilitaire UIC qui va

générer automatiquement les classes `Ui::TPClass` et `Ui_TPClass` dans le fichier `ui_tp.h`. La classe `Ui::TPClass` est instanciée comme donnée membre de la classe principale de votre projet TP. Tout cela se retrouve sur le diagramme de classes suivant :



## Exercice 2 Création de l'interface utilisateur

Il se peut que l'auto-complétion de Visual Studio ne fonctionne pas et que les noms de classes Qt soit soulignées en rouge. Pour résoudre ce bug, compilez le projet puis choisissez le menu **Projet / Relancer l'analyse de la solution**.



Ce que vous obtiendrez à la fin de cet exercice

Pour bien comprendre le fonctionnement objet de Qt, nous allons exceptionnellement nous passer de Qt Designer.

- ☐ Munissez la classe `TP` d'un membre de type `QPushButton*`.
  - ➡ Dans le constructeur, instanciez ce `QPushButton` en passant à son constructeur un pointeur vers la fenêtre principale. De ce fait, le `QPushButton` devient un enfant de la fenêtre principale et sera automatiquement détruit à la destruction de la fenêtre principale.

Nous devons maintenant afficher ce **QPushButton** dans la fenêtre. Cela se fait en utilisant un « layout ».

- 📄 Dans le constructeur, créez dynamiquement un layout (par exemple un **QHBoxLayout**)
- 📄 Ajoutez le **QPushButton** au layout à l'aide de la méthode **addWidget** du layout.
- 📄 Affectez le layout à la fenêtre principale à l'aide de la fonction **setLayout** de la fenêtre.
  - ➡ Compilez et exécutez, vous devriez obtenir une fenêtre avec en son centre un bouton vide sur toute la largeur.

Nous voudrions afficher une boîte de dialogue affichant le message « Félicitations ! Vous avez réussi à me cliquer ! » lors du clic sur ce bouton. Lorsqu'un bouton est cliqué, il émet le signal **clicked()**. Nous allons donc créer un slot pour recevoir ce signal et établir la connexion.

- 📄 Ajoutez à la classe **TP** un slot privé nommé **clic**.
  - ➡ Sa signature sera « **void clic();** »
  - ➡ L'affichage de la boîte de dialogue peut se faire à l'aide de la classe **QMessageBox** :

```
QMessageBox::information(this, "Bravo", "Félicitations ! Vous avez réussi à me cliquer !");
```

- 📄 Dans le constructeur, réalisez la connexion entre le signal **clicked()** du bouton et le slot **clic()** de la fenêtre.
  - ➡ Compilez et exécutez, vous devriez voir le message de félicitation lors du clic sur le bouton.

### Exercice 3 Compte à rebours

Nous souhaitons que l'utilisateur n'ait que 5 secondes pour appuyer sur le bouton, et que le bouton affiche le compte à rebours. Pour cela, nous allons utiliser un **QTimer**.

📄 Google translate de la [documentation de la classe QTimer](#) : La classe **QTimer** fournit une interface de programmation de haut niveau pour les temporisateurs. Pour l'utiliser, créer une **QTimer**, connectez son signal **timeout()** au slots appropriés, et appeler **start()**. Dès lors, il va émettre le signal **timeout()** à des intervalles constants.

- 📄 Munissez la classe **TP** d'une variable entière **m\_nRemainingTime** qui sera le nombre de secondes restantes pour pouvoir appuyer sur le bouton.
  - ➡ Initialisez-là à 6 dans le constructeur.
- 📄 Ajoutez à la classe **TP** un slot privé nommé **tic**.
  - ➡ Dans ce slot, vous définirez le texte du bouton par le code suivant :

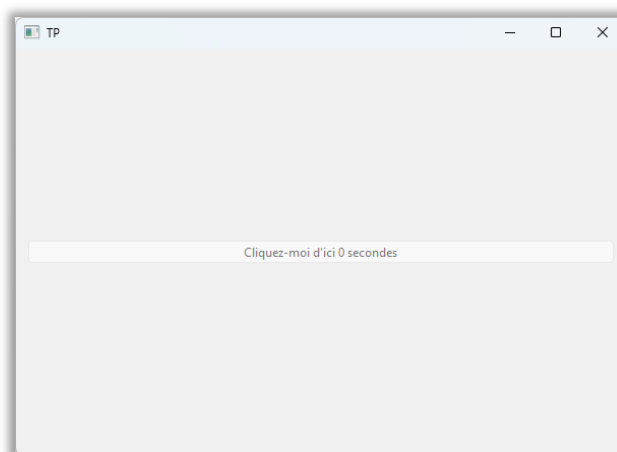
```
m_nRemainingTime--;  
m_theButton->setText(QString("Cliquez-moi d'ici %1 secondes").arg(m_nRemainingTime));
```

- 📄 Munissez la classe **TP** d'un pointeur vers **QTimer**
  - ➡ Dans le constructeur, instanciez ce **QTimer** en passant à son constructeur un pointeur vers la fenêtre principale. De ce fait, le **QTimer** devient un enfant de la fenêtre principale et sera automatiquement détruit à la destruction de la fenêtre principale.
  - ➡ Connectez son signal **timeout()** au slot **tic()**.
  - ➡ En fin de constructeur, démarrez le timer pour une période de 1 seconde par sa méthode **start()** et appelez une fois la méthode **tic()** pour initialiser le texte du bouton.
- 📄 Compilez et exécutez, vous devriez obtenir un compte à rebours sur le bouton.

## Exercice 4 Fin de la partie à 0

Nous souhaitons qu'arrivé à 0, le compte à rebours s'arrête et que le bouton se désactive. Par ailleurs, si le bouton a été cliqué dans l'intervalle des 5 secondes, nous souhaiterions que le message « Vous avez gagné » s'affiche. Dans le cas contraire, ce sera le message « Vous avez perdu ». Le clic sur le bouton ne devra maintenant plus afficher le message de félicitation.

- 📄 Supprimez le message de félicitation du slot **clic()**.
- 📄 Munissez la classe **TP** d'un membre booléen valant **false** par défaut et passé à **true** lors du clic sur le bouton.
- 📄 Ajoutez un nouveau slot nommé **end()** à la classe **TP**.
  - ➡ Ce slot arrêtera le timer par sa méthode **stop()**.
  - ➡ Ce slot désactivera le bouton grâce à sa méthode **setEnabled()**.
- 📄 Ajoutez à la classe **TP** un signal **NoMoreTime(bool)**
  - ➡ Emettez ce signal lorsque le temps restant est arrivé à 0.
    - ➡ Le booléen à transmettre est le booléen permettant de savoir si le bouton a été cliqué.
- 📄 Dans le constructeur de la classe **TP**, connectez le signal **NoMoreTime(bool)** au slot **end()**.
  - ➡ Remarquez que vous pouvez connecter un signal muni de paramètres à un slot qui en a moins (voire pas du tout).
- 📄 Compilez et exécutez, le bouton devrait se griser lorsque le compteur arrive à 0.



Ce que vous devriez obtenir à ce point

L'affichage du message « Vous avez gagné » ou « perdu » sera géré par un autre objet Qt, indépendant de la fenêtre principale.

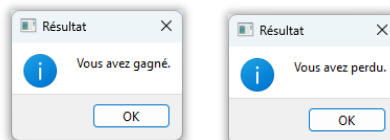
- 📄 Ajoutez au projet une nouvelle classe en suivant cette procédure :
 

<ul style="list-style-type: none"> <li>➡ Menu <b>Projet / Add Qt Class...</b></li> <li>➡ Type de classe : <b>Qt Class</b></li> <li>➡ Class name : <b>QControler</b></li> <li>➡ <b>Add → Next → Finish.</b></li> </ul>	<ul style="list-style-type: none"> <li>➡ Clic droit sur l'icône du projet <b>TP</b> puis <b>Add New...</b></li> <li>➡ <b>C++ Class → Choose...</b></li> <li>➡ Class name : <b>QControler</b></li> <li>➡ Base class : <b>QObject</b></li> <li>➡ <input checked="" type="checkbox"/> <b>Include QObject</b></li> <li>➡ <input checked="" type="checkbox"/> <b>Add Q_OBJECT</b></li> <li>➡ <b>Suivant → Terminer</b></li> </ul>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- Munissez la classe **QController** d'un slot public nommé **end()** prenant en paramètre un booléen. L'objet de ce slot est d'être appelé lorsque le compteur arrive à 0. Le booléen vaudra **true** si le bouton a été cliqué.
- Le code de ce slot est le suivant (en supposant que le paramètre booléen soit nommé **bClicked**)

```
QMessageBox::information(nullptr, "Résultat", QString("Vous avez %1.").arg(bClicked ? "gagné" : "perdu"));
```

- Dans la fonction **main**, créer une instance de **QController**, puis associez le signal **NoMoreTime(bool)** de la classe **TP** au slot **end(bool)** du **QController**.
- Compilez et exécutez
  - Au bout des 5 secondes, vous devriez voir le message « Vous avez gagné » ou « Vous avez perdu » selon que vous ayez appuyé sur le bouton ou non.



## Exercice 5 Donner la possibilité de rejouer

Le nom de la classe **QController** prend maintenant tout son sens car cette classe va contrôler le « jeu ». Lorsque la partie est finie, au message « Vous avez perdu » / « Vous avez gagné » on doit maintenant ajouter la question « Voulez-vous rejouer ? ». Si le joueur répond oui, alors on détruit la fenêtre principale et on la recrée. Cela a pour conséquence de relancer la partie.

- Dans la fonction **main**, supprimez la création de la fenêtre principale.
- Munissez la classe **QController** d'un pointeur intelligent vers un **TP**.
  - Vous devez l'instancier dans le constructeur. **TP** étant un **QWidget** et **QController** étant un **QObject**, vous ne pouvez pas les associer par les constructeurs
    - N'oubliez pas de connecter les signaux et slots nécessaires
    - N'oubliez pas d'afficher l'objet **TP**.
- Modifiez le slot **end(bool)** de façon à pouvoir prendre en compte la réponse de l'utilisateur.
  - Ce bout de code pourra vous être utile :

```
QMessageBox::StandardButton ans = QMessageBox::question(nullptr,
    "Résultat",
    QString("Vous avez %1.\nVoulez vous rejouer ?")
    .arg(bClicked ? "gagné" : "perdu"));
if(ans == QMessageBox::Yes)
{
    //Réinitialiser le jeu
}
else
{
    //Supprimer l'objet TP pour quitter le jeu.
}
```