


2 | TP 2 : horloge, signaux, premiers widgets

Remarque : n'hésitez pas à lire la documentation sur `doc.qt.io`, ou l'aide en ligne de QtCreator (sélectionner un mot et ).

On souhaite créer une pendule pour jouer aux échecs¹. Ces pendules ont une particularité : elles contiennent 2 horloges qui servent à décompter le temps de chaque joueur, on bascule d'un joueur à l'autre en appuyant sur un seul bouton, ce qui arrête une horloge et démarre l'autre.

2.1 Horloge seule

On souhaite créer une seule horloge, i.e. un temps est donné au début de l'application, ce temps est diminué à chaque seconde, lorsque le temps est nul l'application s'arrête.

Des *widgets* permettront de visualiser ces informations. Voilà l'interface minimale à construire :

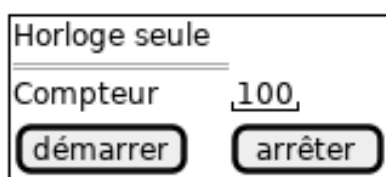


FIGURE 2.1 – Vue schématique de l'interface minimale

On va utiliser **QtCreator** pour faciliter l'écriture et l'insertion de widget :

1. créer un nouveau projet (+ Nouveau projet), cf figure 2.2

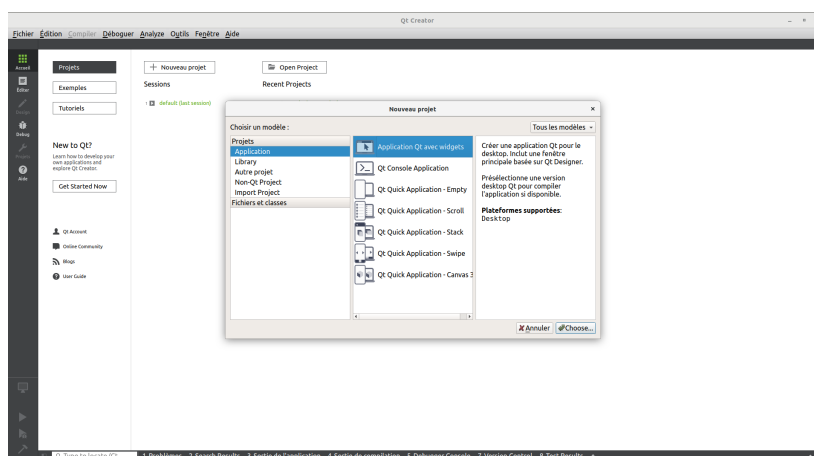


FIGURE 2.2 – Nouveau projet QT

2. après avoir choisi (ou pas) le chemin du projet, et le kit de développement - a priori QT 5.XX, pour PC, on déclare ses fichiers de développement. On modifie ici : le nom du widget principal `horlogeSeule`, le type `Widget` (et pas `MainWindow`!) et pas d'UI ("*ne pas générer le fichier d'interface*") pour l'instant.
3. à ce stade, vous devez donc avoir les informations de la figure 2.3.
4. une fois le projet créé, **QtCreator** a créé l'arborescence et le `.pro` du projet (voir figure 2.4a).
5. si vous compilez et exécutez (CTL-R), vous avez une magnifique... fenêtre vide (hourra, votre 1ère application graphique!), comme le montre la figure 2.4b.

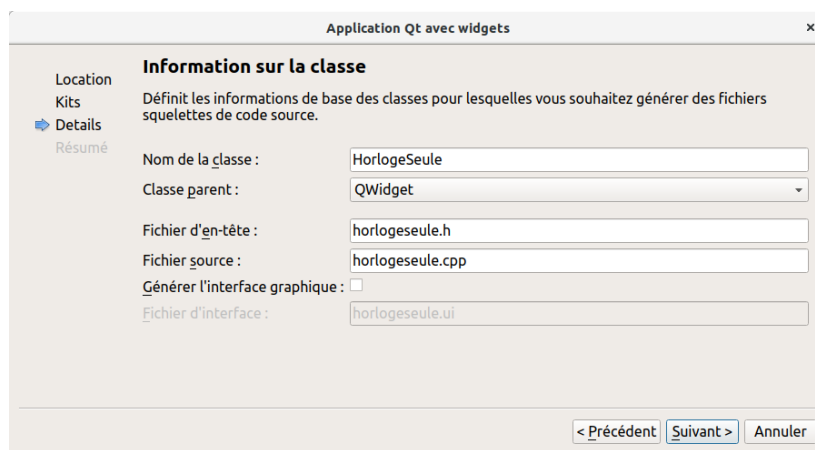
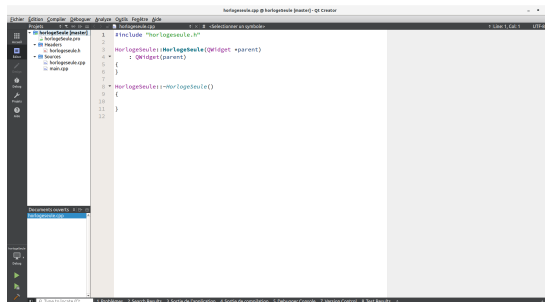
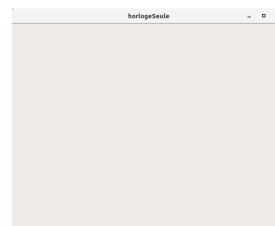


FIGURE 2.3 – Création de la classe HorlogeSeule



(a) Projet QT de base



(b) Exécution de l'application de base...

Note : si vous ne l'avez pas vu, ouvrez le fichier `main.cpp`, une `QApplication app` est créée (donc graphique, par rapport au TP1) et une `HorlogeSeule w` aussi, comme c'est un widget (cf. la création du projet), on peut faire `w.show()`. Tout cela est défini par QT.

On va maintenant ajouter les widgets qui ressemblent à l'interface prévue :

1. un texte « Compteur »
2. un champ avec la valeur du compteur (100 par défaut)
3. un bouton « Démarrer »
4. un bouton « Arrêter »

Pour cela on change les membres du widget `horlogeSeule+`, dans le `.h` :

```
1 #include <QLabel>
2 #include <QPushButton>
3
4 class HorlogeSeule : public QWidget
5 {
6     Q_OBJECT
7 private:
8     unsigned compteur;
9
10 public:
11     HorlogeSeule(QWidget *parent = 0);
12     ~HorlogeSeule();
13
14     QLabel * texte;
15     QLabel * valeur;
16
```

1. https://fr.wikipedia.org/wiki/Pendule_d'%C3%A9checs

```

17     QPushButton *start, *stop;
18 };

```

On ajoute donc :

- un QLabel pour le texte « Compteur : », texte fixe qui ne sera pas modifié,
- un autre QLabel pour la valeur du compteur, qui évoluera,
- 2 QPushButton « start » et « stop »

Note : il faut inclure les fichier de QT correspondant :

```

1 #include <QLabel>
2 #include <QPushButton>

```

Note2 : on inclut les sous-widget (QLabel, QPushButton) en tant que pointeurs, pour diminuer l’empreinte mémoire du widget.

Note3 : `compteur` est une variable (entière) `private` de la classe `HorlogeSeule`. Seul ce widget peut en changer la valeur directement.

Le fichier de déclaration de la classe contient l’instanciation de ces variables, dans le constructeur :

```

1 HorlogeSeule::HorlogeSeule(QWidget *parent)
2     : QWidget(parent)
3 {
4     compteur = 100;
5
6     texte = new QLabel("Compteur_:", this);
7
8     valeur = new QLabel(this);
9     valeur->setNum((int)compteur);
10
11     start = new QPushButton("Démarrer", this);
12
13     stop = new QPushButton("Arrêter", this);
14 }

```

Si on compile/exécute, cela rend :

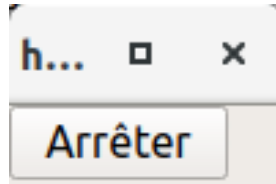


FIGURE 2.4 – Ce n’est pas réussi...

On ne peut pas dire que cela soit pratique ! En fait, tous les sous-widgets se sont superposés, il faut les ranger. Pour cela, on utilise un *layout*. On va ranger les sous-widgets dans un `QVBoxLayout` (il faut inclure `#include <QVBoxLayout>`).

Dans le constructeur on va donc ajouter :

```

1 <...instanciation des sous-widgets...>
2
3     QVBoxLayout * vLayout = new QVBoxLayout(this);
4     vLayout->addWidget(texte);
5     vLayout->addWidget(valeur);
6     vLayout->addWidget(start);
7     vLayout->addWidget(stop);
8
9     setLayout(vLayout);

```

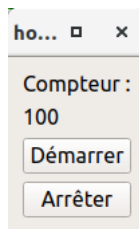
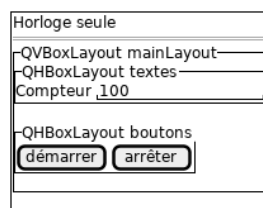


FIGURE 2.5 – Ça marche pas trop bien

Et voilà ! Ah mince ! Les deux boutons ne sont pas alignés côte à côte, « Compteur » et « 100 » non plus.

Pas de problème, on met les 2 boutons dans un *layout* horizontal (`QHBoxLayout`, avec l'inclusion `#include <QHBoxLayout>`), lui-même dans le `QVBoxLayout`. On fait pareil pour les 2 `QLabel`.

Cela revient donc à construire les inclusions de *layouts* suivants :



Hourra (bis), voilà notre interface graphique construite !

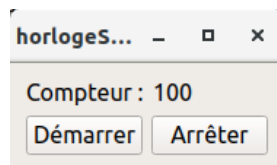


FIGURE 2.6 – Là c'est mieux

Les sources de toutes ces étapes sont dans le fichier fourni pour comparer avec votre version (fichier "horlogeSeule.zip"). Votre travail à partir de là :

1. utiliser un `QTimer` (cf annexe 2.3) pour décrémenter chaque seconde le compteur,
2. commencer le décompte du compteur lorsque l'on appuie sur `Démarrer` ,
3. mettre en pause le décompte lorsque l'on appuie sur `Arrêter` ,
4. afficher un message « Décompte terminé » lorsque le compteur atteint 0, puis fermer l'application.

Évidemment, il faut mettre en place des messages...

2.2 Version 2 : 2 horloges

Une fois que le *widget* `HorlogeSeule` fonctionne, on peut ré-utiliser ce composant. On va donc instancier 2 `HorlogeSeule` (cf source `penduleEchec0`).

```
1 HorlogeSeule horloge0;
2 HorlogeSeule horloge1;
3
4     horloge0.show();
5     horloge1.show();
6
7     QMessageBox msgBox;
8     msgBox.setText("Le décompte est arrivé à sa fin");
9
10    QObject::connect(&horloge0, SIGNAL(finDecompte()), &msgBox, SLOT(exec()))
11    ;
12    QObject::connect(&horloge0, SIGNAL(finDecompte()), &app, SLOT(quit()));
```

Et cela ne marche pas...

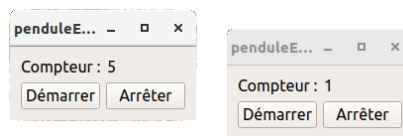


FIGURE 2.7 – 2 composants en vrac

Évidemment, les 2 *widgets* `HorlogeSeule` ne sont pas reliées graphiquement dans une seule fenêtre, rien ne l'a demandé. Il y aura aussi un souci à régler, la fenêtre de message `msgBox` n'est reliée à rien, et les `connect` ne sont actifs que sur l'instance `horloge0`, c'est nul ! Il n'y a eu aucun effort de conception.

En fait, il faut déjà penser que ces 2 *widgets* doivent être dans un *layout* qui permette l'arranger horizontalement, donc dans un seul *widget*. On doit donc commencer par créer un nouveau *widget*, et donc un nouveau couple `.h/.cpp`, appelons-le `penduleEchec`. Toujours la même méthode, sur le projet à gauche dans *QTCreator*, « ajouter un nouveau... Class C++... »

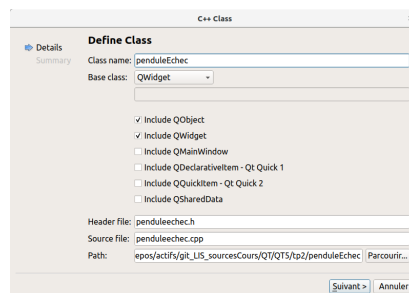


FIGURE 2.8 – Classe Pendule

Dans cette classe, il faut :

1. inclure `"horlogeSeule.h"`
2. ajouter 2 instances d'`HorlogeSeule` (en pointeur donc, et qu'il faudra allouer dans la construction du pendule).

On a donc les sources suivantes :

```
1 #include "horlogeseule.h"
2
3 class PenduleEchec : public QWidget
```

```

4 {
5     Q_OBJECT
6 private:
7
8     HorlogeSeule * horloge0;
9     HorlogeSeule * horloge1;
10
11 public:
12     explicit PenduleEchec(QWidget *parent = nullptr);
13
14 signals:
15
16 public slots:
17 };

```

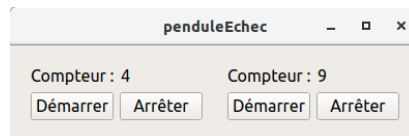
Et le constructeur de « PenduleEchec » :

```

1 PenduleEchec::PenduleEchec(QWidget *parent) : QWidget(parent)
2 {
3     QHBoxLayout * penduleLayout = new QHBoxLayout();
4
5     horloge0 = new HorlogeSeule(this);
6     horloge1 = new HorlogeSeule(this);
7
8     penduleLayout->addWidget(horloge0);
9     penduleLayout->addWidget(horloge1);
10
11     setLayout(penduleLayout);
12 }

```

Hourra, cela fonctionne on a bien nos 2 *widgets* d'horloge !



Ces sources sont disponibles dans « **penduleEchec.zip** ». Il vous reste maintenant à implémenter :

1. pour l'instant les signaux **finDecompte()** ne sont pas pris en compte. Pour que cela fonctionne, il faut que ces signaux soient gérés maintenant par le *widget* **penduleEchec**. Il faut donc créer un *slot* dans ce dernier, qui lance la **msgBox** lorsque l'une ou l'autre des horloges (via leur compteur respectif, arrive à 0). Une schématisation de ces séquences est donnée à la figure 2.9.

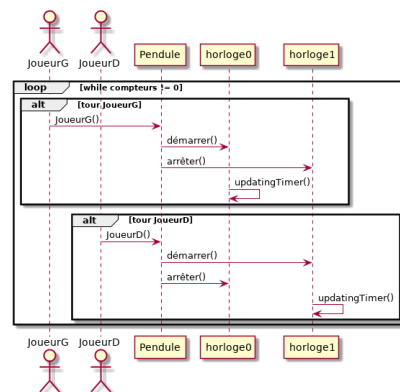


FIGURE 2.9 – Séquences des messages entre les Horloge et le Pendule

- la liaison des 2 horloges, pour l'instant chaque bouton {Démarrer, Arrêter} est lié à « son » horloge. Notre pendule doit agir comme une bascule d'un joueur à l'autre. On rajoute donc 2 boutons « Joueur Gauche » et « Joueur Droit » - ou 1 et 2 comme vous voulez. Ce seront ces boutons qui déclencheront l'arrêt d'une horloge et le (re)démarrage de l'autre (via les signaux horloge->arrêter() et horloge->démarrer()).
- les 2 précédents boutons {Démarrer, Arrêter} de chaque horloge ne servent plus, on peut les conserver comme information pour connaître visuellement l'état de chaque horloge. On va utiliser la propriété `setEnabled(bool)` des boutons (dans la documentation de `QPushButton`, et surtout `QAbstractPushButton`). Lorsque le signal `démarrer()` d'une horloge est reçu :

```
1 void HorlogeSeule::démarrer()
2 {
3     timer->start(1000);
4     start->setEnabled(true);
5     stop->setEnabled(false);
6 }
```

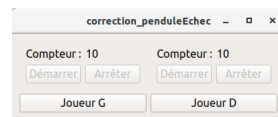
Et inversement pour le signal arrêter. On travaille ici au niveau du composant (« horloge-Seule ») plutôt que du *widget* « PenduleEchec ».

Remarque : en allant regarder les propriétés de `QAbstractButton`, on peut également, à la construction du bouton, ajouter un :

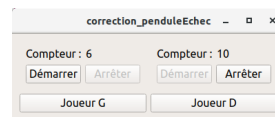
```
1 start->setDisabled(true);
2 start->setCheckable(false);
```

Le `setCheckable(false)` permet d'avertir que ce bouton ne déclenchera plus de signaux `clicked()`. Il faut également déconnecter les signaux `clicked()` de ces boutons, ils ne servent plus qu'à informer l'utilisateur.

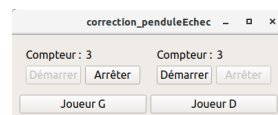
Finalement l'application devrait afficher les écrans suivants :



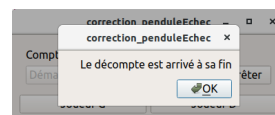
(a) Initialisation de l'application



(b) Joueur gauche a commencé



(c) Au tour de Joueur droit



(d) Une des horloge s'est terminée, l'application s'arrête

2.3 Annexe : QTimer

QT intègre une gestion des alarmes, il faut déclarer un *timer* et son intervalle de rafraîchissement. Et ensuite, comme d'habitude, lier le message d'alarme (fin d'intervalle, `timeout` du `QTimer`) avec la fonction qui permet de gérer ce message.

```
1 //à mettre dans la classe horloge
2 #include <QTimer>
3
4 timer = new QTimer(this);
5 connect(timer, SIGNAL(timeout()), this, SLOT(updatingTimer()));
```