

# TP 26 –Graphismes dans Qt

Comme pour le TP 25, la solution de ce TP est disponible par le gestionnaire de suivi de version Git via le dépôt distant [git@gitlab-lepuy.iut.uca.fr:TPs-QualiteDeDev/TP26.git](https://git@gitlab-lepuy.iut.uca.fr:TPs-QualiteDeDev/TP26.git). Des étiquettes vous permettent de naviguer dans le dépôt parmi les différents exercices.

## Objectif du TP

Vous allez créer un widget qui affiche des balles qui rebondissent, puis intégrer ce widget dans une application.

## Exercice 0 Récupération des sources initiales et création d'un nouveau projet

- 📁 Clonez le dépôt [git@gitlab-lepuy.iut.uca.fr:TPs-QualiteDeDev/TP26.git](https://git@gitlab-lepuy.iut.uca.fr:TPs-QualiteDeDev/TP26.git).
  - ➡ Vous récupérez ainsi un simple répertoire **BouncingBall** contenant des fichiers qui vous seront nécessaires plus tard.
- 📁 Dans ce répertoire, créez un nouveau projet Qt en suivant la même procédure que pour le TP précédent et dont la fenêtre principale hérite d'un **QMainWindow**.
  - ➡ **cc** Sous Qt Creator, faites un clic droit dans le Designer sur la fenêtre éditée et choisissez **Ajouter une barre d'outils**.

👤 Par la suite, nous supposerons que vous avez nommé la classe correspondant à la fenêtre principale « **TP** »





👤 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**.

## Exercice 1 Création d'un widget personnalisé

### 1.1 Création de la classe du widget






- 📁 Ajoutez à votre projet une nouvelle classe Qt
  - ➡ Pour cela, **Projet / Add Qt Class...** et choisissez le modèle **Qt Class** ou **cc** faites un clic droit sur le projet puis choisissez **Add New... / Qt / C++ Class**.
    - ➡ Nommez la classe **QBallsWidget**
    - ➡ Faites-la hériter de **QWidget**
    - ➡ **Projet** Sélectionnez la signature du constructeur « **QWidget\* parent** »
    - ➡ Laissez cochée la case Insert **Q\_OBJECT**
    - ➡ Validez.

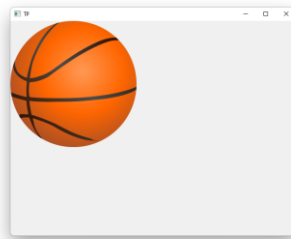
Dans un premier temps, nous allons travailler uniquement sur cette classe. La fenêtre principale n'entre pas encore en jeu. De façon que votre widget soit affiché, il faut modifier le programme principal.

-  Dans la fonction main, commentez la création d'un objet de type **TP** et remplacez-le par la création d'un objet de type **QBallWidget**.
-  Compilez et exécutez. Vous devez obtenir une fenêtre vide.

## 1.2 Personnalisation de son affichage





Pour tester vos compétences en affichage, nous n'allons pour le moment qu'afficher une balle à l'écran.

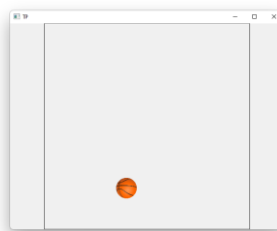
-  Mettez en place le mécanisme de dessin graphique dans la classe **QBallWidget**
  -  Ceci a été vu et fait en cours...
-  Afficher la balle fournie avec ce sujet (fichier **BouncingBall\res\ball.png**) à la position 0,0.
  -  Il vous est conseillé de placer le fichier dans les ressources de l'application (vu en cours...)
  -  Vous devriez obtenir l'affichage suivant :



Votre premier dessin.

## 1.3 Définition du système de coordonnées

-  Faites en sorte que le système de coordonnées de la fenêtre soit orthonormé présentant au moins 100 unités de large et 100 unités de haut, centré dans la fenêtre si la fenêtre n'est pas carrée. L'origine devra se trouver dans le coin inférieur gauche avec l'axe des y dirigé vers le haut.
  -  Pour tester ce système de coordonnées, affichez la balle avec un diamètre de 10 et centrée à la position 40,20.
  -  Dessiner un cadre noir tout autour du système (un rectangle à la position 0,0 de taille 100,100)
  -  Vous devriez obtenir l'affichage suivant :



Test du système de coordonnées.

## 1.4 Faire rebondir la balle

Pour que vous ne perdiez pas de temps avec la simulation physique, une classe l'encapsulant vous est fournie dans les fichiers **BouncingBall.h** et **BouncingBall.cpp**. Son utilisation est détaillée dans l'aide en ligne stockée dans le répertoire **BouncingBall\doc**

Ajoutez à votre projet les fichiers **BouncingBall.h** et **BouncingBall.cpp**.

Cette classe gère la simulation physique d'une balle rebondissante.

Munissez votre widget d'un membre de type **CBouncingBall**.

À sa construction, initialisez-le avec les informations suivantes :

- ➡ Rayon = 5
- ➡ Position initiale : 50,100
- ➡ Vitesse initiale : 100,0
- ➡ Poids : 150
- ➡ Dimension du monde 100

Pour actionner la simulation, il faut appeler la méthode **step()** régulièrement. À chaque appel, la position et la rotation de la balle sont modifiées pour respecter le modèle physique implémenté. Pour faire une animation, il faut activer la méthode **step()** à l'aide d'un timer.

Mettez en place un timer qui s'active le plus rapidement possible (un intervalle nul permet d'activer le timer dès que possible)

Vous pouvez également utiliser la méthode **startTimer()** des **QObject** et surcharger la méthode **timerEvent** (RTFM).

À chaque « timeout », appelez la méthode **step()** de la balle puis la méthode **update()** du widget pour mettre à jour son affichage.

Réalisez l'affichage de la balle

Affichez la balle à la position fournie par ses méthodes **x()** et **y()** (n'oubliez pas de la centrer sur cette position).

Faites tourner la balle de l'angle fourni par la méthode **angle()** de l'objet **CBouncingBall**.


Attention, l'angle fourni par la méthode **CBouncingBall::angle()** est exprimé en radian, alors que les angles des méthodes **rotate** de Qt doivent être exprimés en degré. Vous pouvez utiliser la fonction **qRadiansToDegrees** de la bibliothèque **<QtMath>** pour réaliser la conversion.

Compilez / exécutez, vous devriez obtenir une balle qui rebondit dans la fenêtre.



Votre première balle rebondissante.

## Exercice 2 Incorporer le widget personnalisé dans une application

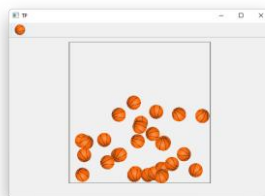
- ❏ Annulez vos modifications de la fonction main de façon à afficher maintenant la fenêtre principale.
- ❏ Ouvrez le fichier **TP.ui** pour personnaliser l'interface à l'aide de Qt Designer.
- ❏ Incorporez au centre de la fenêtre un simple Widget (outil  Widget )

🧑 Nous devons spécifier à Qt Designer que ce widget est en réalité votre widget personnalisé de type **QBallswidget**

- ❏ Sélectionnez le widget nouvellement incorporé dans la fenêtre et faites un clic droit dessus.
  - ➡ Choisissez *Promouvoir en...*
  - ➡ Dans *nom de la classe promue*, spécifiez « **QBallswidget** »
  - ➡ Fichier d'en-tête « **qballswidget.h** »
  - ➡ Cliquez sur *Ajouter*
  - ➡ Cliquez sur *Promouvoir*
- ❏ Enregistrez le fichier, compilez et exécutez.
  - ➡ La balle rebondissante doit rebondir dans la fenêtre principale (vous devez voir la barre d'outils en haut de la fenêtre)

## Exercice 3 Ajouter d'autres balles

- ❏ Modifiez la classe **QWidgetsBalls** de la façon suivante
  - ➡ Il n'y a plus le membre correspondant à une balle
  - ➡ Ajoutez un membre pour gérer une liste de balles
  - ➡ Ajoutez un slot public **addBall()** qui ajoutera une balle à la liste. La nouvelle balle ajoutée aura les caractéristiques suivantes :
    - ➡ Rayon : 5
    - ➡ Position aléatoire comprise en x et y entre 5 et 95
    - ➡ Une vitesse initiale aléatoire en x comprise entre -100 et +100.
    - ➡ Une vitesse initiale aléatoire en y comprise entre -50 et +50.
    - ➡ Un poids aléatoire compris entre 100 et 200
    - ➡ Dimension du monde 100
  - ➡ Modifiez le timer et l'affichage de façon à mettre à jour et afficher toutes les balles.
- ❏ Munissez l'interface graphique d'une action dans la barre d'outils et connectez son signal **triggered()** à un slot **addBall()** que vous venez d'ajouter à votre classe **QBallswidget**.



Version finale

🧑 Une version optimisée et munie d'une mesure d'images par seconde est disponible dans le dépôt Git.