TP3 : Fenêtre principale, agencement des fenêtres dans une interface, dialogue et tracé régulateur des fenêtres

Objectif : découvrir comment personnaliser et agencer vos fenêtres avec la bibliothèque Qt, les quelques règles ergonomiques qui conditionnent cette disposition. Beaucoup d'éléments seront à reprendre pour votre rapport de projet d'ACSI-BDD .

1. Ergonomie : tracé régulateur

On dispose les informations sur une fenêtre en fonction de la *Caractéristiques de visibilité et d'accessibilité d'un écran (voir résumé ergonomie)*. Cependant **la position des informations doit être cohérente** pour l'homogénéité de l'ensemble.

On utilise un **tracé régulateur** (terme d'architecture désignant un schéma type dont le rôle est de préciser le positionnement de divers éléments sur une surface) . Il détermine l'agencement des fenêtres de l'application.

schéma :exemple de tracé régulateur qui permet d'homogénéiser les différentes fenêtres de l'application

<table>
<tr><td colspan="2">commandes</td></tr>
<tr><td>Zone de sélection</td><td>Zone de travail</td></tr>
<tr><td colspan="2">Zone de message à lecture optionnelle</td></tr>
</table>

Il faut organiser les zones de manipulation dans le sens de la lecture et en fonction de la fréquence d'utilisation. Ergonomie : tracé régulateur

schéma 4 : exemple d'agencement de zone de manipulation

barre de menu d'outils

<table>
<tr><td>commandes importantes</td><td>commandes fréquemment utilisées</td><td>commandes annexes</td></tr>
</table>

palette

<table>
<tr><td>outils importants</td></tr>
<tr><td>outils fréquemment utilisés</td></tr>
<tr><td>outils annexes</td></tr>
</table>

Ces commandes devront être conçues de la façon la plus compacte possible afin que l'utilisateur ait toutes les commandes sous les yeux et que les déplacements de la souris soient limités. **Il faut regrouper les informations.**

2. Fenêtre sur QT

Il existe 3 types de fenêtre sous Qt. Les fenêtres de type « QDialog » ,de type « QMainWindow » et de type « QWizard ».
Qdialog est une fenêtre classique : pour informer l'utilisateur, récupérer des informations qui seront validées par un bouton. C'est une fenêtre simple à construire.
« QMainWindow » est une fenêtre principale, c'est celle que nous allons le plus étudier.
« QWizard » est un ensemble de fenêtres de type « QDialog » utiles pour l'installation d'un logiciel ou pour exécuter un ensemble de tâches successives.

○ *Fenêtre de dialogue*

C'est une fenêtre qui permet de consulter ou modifier un ensemble de données, avec des boutons qui servent à déclencher le traitement (exemple : fenêtre d'impression).

Elles sont faciles à comprendre mais sont adaptées à des tâches structurées car elles imposent une certaine rigidité.

Recommandations :

Présenter uniquement les informations pertinentes vis-à-vis de la tâche

Présenter les composants dans l'ordre d'utilisation

Guider l'utilisateur

Minimiser le déplacement de la souris (prévoir une saisie au clavier, car c'est plus rapide pour les utilisateurs expérimentés), **permettre un accès rapide et direct aux utilisateurs expérimentés.**

Faciliter l'accès aux composants de la fenêtre les plus fréquemment utilisés.

Mettre en évidence les éléments les plus importants.

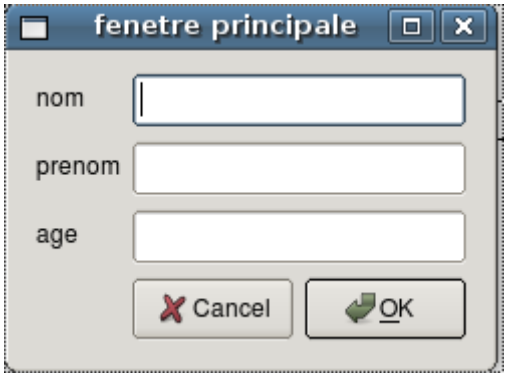
Regrouper les commandes en fonction de leur sens ou de l'objet auquel elles se rapportent.

Uniformiser les termes et la syntaxe employés pour toutes les fenêtres de l'application.

Éviter les fenêtres trop verbeuses. (faire un tri dans les informations si la fenêtre est trop riche et trouver une autre solution de présentation)

Exercice 1

On utilise le code ci dessous (que vous compléterez) pour réaliser une fenêtre « QDialog » de base :



Cet exemple met en oeuvre l'utilisation de 2 « QDialog », une fenêtre classique dérivée et une fenêtre simplifiée (« QMessageBox » pour demander confirmation lorsque l'on quitte) qui se ferme avec une façon plus correct que précédemment.

```
#!/usr/bin/env python
#-*- coding:utf8 -*-
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class DIALOG_Fenetre(QDialog):
    def __init__(self, parent=None):
        super( DIALOG_Fenetre, self).__init__(parent)
# déclaration de 1 objets layout de type QGridLayout de nom : layout
# déclaration de 3 QLabel de noms : Label_nom Label_prenom Label_age
# déclaration de 3 QLineEdit de noms : nom prenom age et qui commence par self
pour être utilisé dans les méthodes (SLOT)

#disposition des objets dans le QGridLayout de façon à être conforme à l'interface

        self.buttonBox = QDialogButtonBox() #boite de boutons
        self.buttonBox.setGeometry(QRect(30,240,341,32))
        self.buttonBox.setOrientation(Qt.Horizontal)
        self.buttonBox.setStandardButtons(QDialogButtonBox.Cancel|QDialogButtonBox.Ok)
        layout.addWidget(self.buttonBox, 3, 1)  #cette ligne peut être modifié
        self.setLayout(layout)
        self.connect(self.buttonBox,SIGNAL("accepted()"),self.accept)
        self.connect(self.buttonBox,SIGNAL("rejected()"),self.fermer)

    def accept(self):
        print self.nom.text(), self.prenom.text(),  self.age.text()
        ecran = QDesktopWidget().screenGeometry() #resolution de l'ecran
        tailleWidget =  self.geometry() #dimension du widget
        self.move((ecran.width()-tailleWidget.width())/2, (ecran.height()-tailleWidget.height())/2)

    def fermer(self):    # genere l'evenement closeEvent
        self.close()

    def closeEvent(self, event):     #traitement de l'evenement closeEvent
        reponse = QMessageBox.question(self, 'Message',
            "Etez vous sur de vouloir quitter?", QMessageBox.Yes, QMessageBox.No)
        if reponse == QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()
```

➔ **Compléter le code ci dessus en respectant les commentaires en bleu (déclaration des objets, disposition des objets dans le layout).**

Exercice 2

Sur le même modèle, créer une fenêtre (QDialog) avec 2 boutons et un bouton pour quitter.

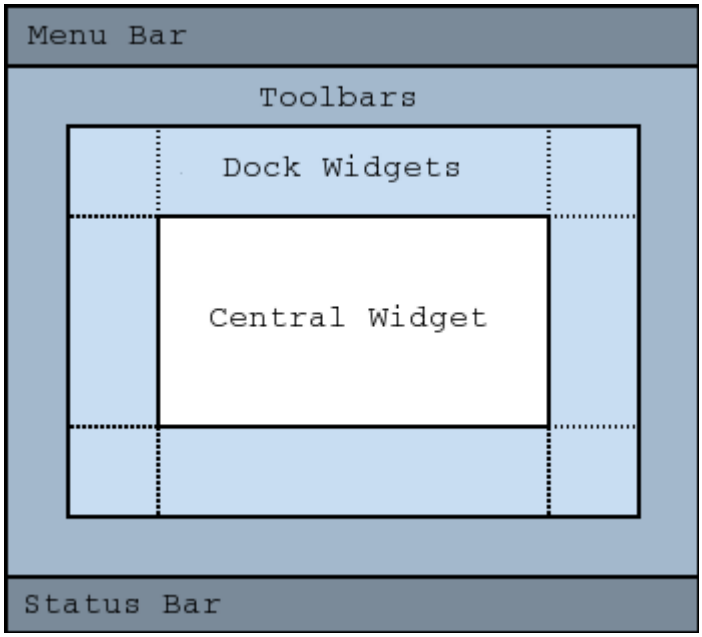
➔ Modifier le code ci dessous : pour ouvrir deux fenêtres, réalisées avec des objets graphiques (Widget) issus d'autres fichiers python, il suffit d'importer les bons fichiers (de votre choix) et d'instancier le bonne objet dans le code ci dessous.

```
#!/usr/bin/env python
#-*- coding:utf8 -*-
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

import exoTP3_1fini       #importation des fichiers python
import exoTP2_1fini      #modification possible du nom des fichiers importés

class DIALOG_Fenetre2(QDialog):
    def __init__(self, parent=None):
        super( DIALOG_Fenetre2, self).__init__(parent)
        self.bouton1 = QPushButton("application1")
        self.bouton2 = QPushButton("application2")
        layout = QGridLayout()
        layout.addWidget(self.bouton1,0,1)
        layout.addWidget(self.bouton2,1,1)

        self.boiteDeBoutons = QDialogButtonBox()
        self.boiteDeBoutons.setGeometry(QRect(30,240,341,32))
        self.boiteDeBoutons.setOrientation(Qt.Horizontal)
        self.boiteDeBoutons.setStandardButtons(QDialogButtonBox.Cancel)
        layout.addWidget(self.boiteDeBoutons, 2, 1)
        self.setLayout(layout)
        self.connect(self.boiteDeBoutons,SIGNAL("rejected()"),self.fermer)
        self.connect(self.bouton1,SIGNAL("clicked()"),self.ouvrirFenetre1)
        self.connect(self.bouton2,SIGNAL("clicked()"),self.ouvrirFenetre2)

    def ouvrirFenetre1(self):
        Fenetre = exoTP3_1fini.DIALOG_Fenetre()  #utilisation des objets des fichiers python importes
                    # autorise une taille plus grande mais pas plus petite
        Fenetre.resize(QSize(QRect(0,0,300,200).size()).expandedTo(Fenetre.minimumSizeHint()))
        Fenetre.show()
        Fenetre.setWindowTitle("tp3 : exercice 1")
        Fenetre.exec_()

    def ouvrirFenetre2(self):
#??????????????????????????  instancier le bon objet
        Fenetre.resize(QSize(QRect(0,0,450,200).size()).expandedTo(Fenetre.minimumSizeHint()))
        Fenetre.show()
        Fenetre.setWindowTitle("tp2 : exercice 1")
        Fenetre.exec_()
```

3. Fenêtre Principale sur QT et Dialogue avec l'utilisateur

- ***Présentation de fenêtre QmainWindow***

éléments qui peuvent constituer une fenêtre principale.



Détails :

- **Menu Bar** : c'est la barre de menus. C'est là que vous allez pouvoir créer votre menu Fichier, Edition, Affichage, Aide, etc.
- **Toolbars** : les barres d'outils. Dans un éditeur de texte, on a par exemple des icônes pour créer un nouveau fichier, pour enregistrer, etc.
- **Dock Widgets** : plus complexes et plus rarement utilisés, ces docks sont des conteneurs que l'on place autour de la fenêtre principale. Ils peuvent contenir des outils, par exemple les différents types de pinceau que l'on peut utiliser quand on fait un logiciel de dessin.
- **Central Widget** : c'est le coeur de la fenêtre, là où il y aura le contenu proprement dit.
- **Status Bar** : la barre d'état. Elle affiche en général l'état du programme (Prêt / Enregistrement en cours, etc.).

on distingue 2 types de QMainWindow :

- Les **SDI (Single Document Interface)** : elles ne peuvent afficher qu'un document à la fois. C'est le cas de Bloc-Notes, emacs
- Les **MDI (Multiple Document Interface)** : elles peuvent afficher plusieurs documents à la fois. Elles affichent des sous-fenêtres dans la zone centrale. C'est le cas par exemple de Qt Designer

Avant de présenter un exemple de réalisation d'une fenêtre principale, commençons par présenter les éléments ergonomiques à prendre en compte pour réaliser cette interface.

- **Ergonomie : Dialogue Homme / machine**

Il désigne les échanges entre l'utilisateur et le logiciel. Il existe différentes formes de modes de dialogue (appui sur les touches du clavier..... manipulation des composants de l'interface). Les limites ne sont pas toujours claires entre ces différentes formes qui se combinent parfois et qui doivent tenir compte de l'évolution de l'utilisateur.

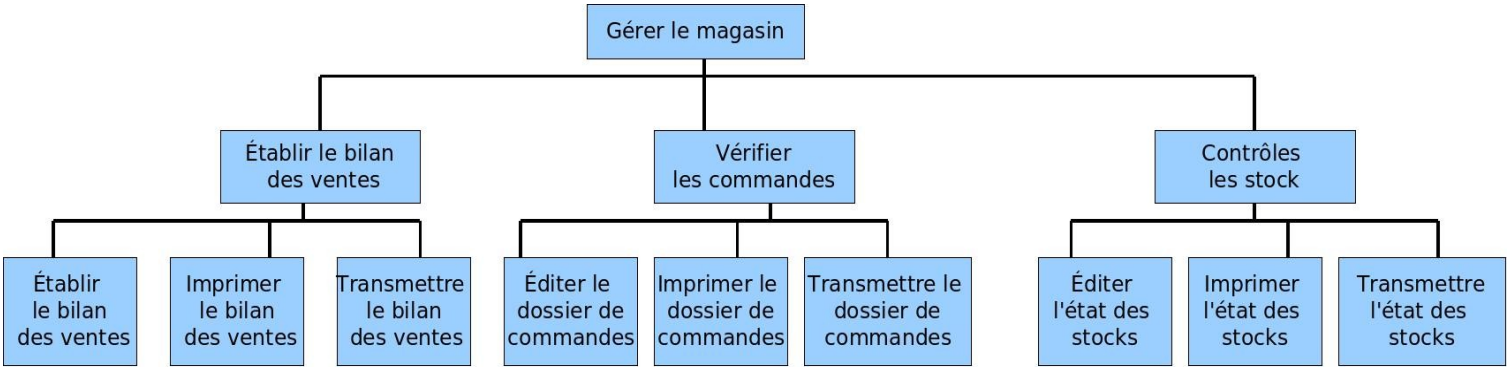
- **Organisation du dialogue**

Il faut organiser la façon dont s'articulent les interventions de l'utilisateur et du logiciel.

conseils :

- Présenter les commandes et les données dans l'ordre d'utilisation. Regrouper les informations relatives à une activité sur la même fenêtre.

schéma 7 : exemple, la gestion d'un magasin se décompose en 3 sous-tâches



- Regrouper les informations relatives à une activité sur la même fenêtre (Chaque fenêtre du logiciel est dédiée à une sous tâche. Elle présente à l'utilisateur les données qui vont lui permettre de prendre la décision requise et les actions à réaliser).
- Laisser l'initiative du dialogue à l'utilisateur (permettre : défaire, annuler)
- Guider l'utilisateur pour faciliter la navigation (une trace du chemin parcouru depuis la fenêtre d'accueil est très utile)

=> **ce diagramme ou un équivalent est obligatoire dans votre rapport**

- **Conception des fenêtres**

La majorité des interfaces homme/machine utilise des fenêtres (zone gérée de manière indépendante) , chaque fenêtre regroupe un ensemble d'objets qui vont servir d'outils de dialogue.

1. le multi-fenêtrage

Deux modes d'affichage sont possibles : **par tuilage**(organisation fixe des fenêtres) ou **par recouvrement** (déplacement des fenêtres possibles).

Le recouvrement est beaucoup plus « souple » pour l'utilisateur (possibilité de comparer des fenêtres, organiser son interface, exécuter plusieurs taches en parallèle ...Très utile pour les utilisateurs expérimentés, il fait gagner du temps [Mayhew 92]).

Recommandations :

L'utilisation des fenêtres ne doit pas perturber la tâche « métier »

Faciliter l'activation et l'ouverture de fenêtres.

Les commandes d'organisation des fenêtres doivent être faciles à mémoriser

Mettre en évidence la fenêtre active

Utiliser le tuilage pour les utilisateurs occasionnels (Web)**, lorsque les sorties sont prévisibles et que l'écran est suffisamment grand** (avec un minimum de barre de défilement).

Autoriser le recouvrement des fenêtres pour les utilisateurs expérimentés, lorsque les sorties ne sont pas prévisibles ou les petits écrans.(espace de travail personnel)

Autoriser la mémorisation des arrangements de fenêtres.

Moins il y a de fenêtres, plus elles sont complexes. Donc augmenter le nombre de fenêtres pour une utilisation peu fréquente.

Diminuer le nombre de fenêtres en cas d'utilisation fréquente (si on les utilise souvent on assimile facilement leur complexité).

Minimiser la quantité d'informations à mémoriser d'une fenêtre à l'autre.

Exercice 3 :Réalisation d'un menu

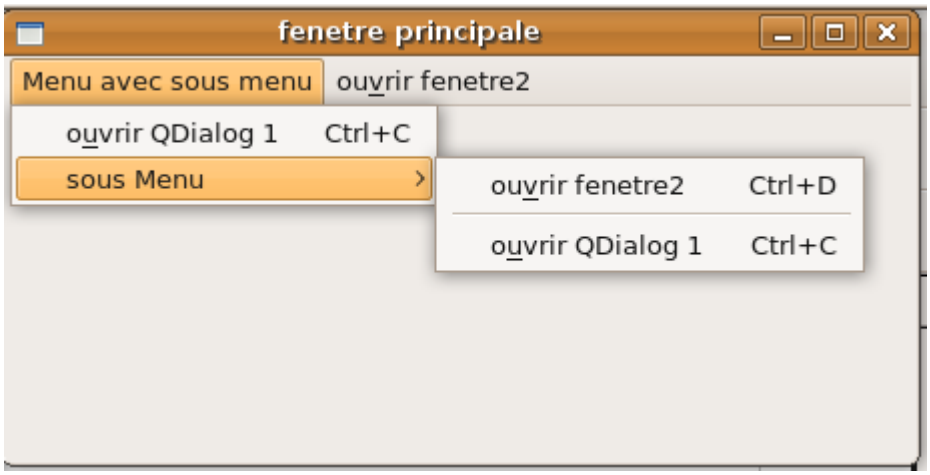
A partir du code dans le fichier texte, vous devez modifier ce code afin d'obtenir l'interface ci dessous

Dans le fichier texte, vous trouvez le code suivant :

```
self.barreDeMenu=self.menuBar()  # constructeur de la barre de menu

 exoTP3_1fini_Action = QAction( "&ouvrir QDialog 1", self) # constructeur d une action
 exoTP3_1fini_Action.setShortcut(QKeySequence("Ctrl+A")) #raccourci pour appeler l'action
helpText = "aide sur le bouton 1"
 exoTP3_1fini_Action.setToolTip(helpText) #texte d'aide pour les barres d'outils
 exoTP3_1fini_Action.setStatusTip(helpText) #texte d'aide pour les barres d'états
#        exoTP3_1fini_Action.setIcon(QIcon(":/image1.ico"))
self.connect( exoTP3_1fini_Action, SIGNAL("triggered()"), self.ouvrirFenetre1)
self.barreDeMenu.addAction(exoTP3_1fini_Action)
```

Qt utilise des actions plutôt qu'une notion de sous menu car dans un menu on peut réaliser la même action à des endroits différents des menus (exemple imprimer).

➔ travail à réaliser :modifier le code joint « exoTP3_3.py » dans le fichier texte pour obtenir la fenêtre suivante (la position des objets est inversée et les raccourcis sont modifiés):



ajout d'icônes :
chercher 2 icônes sur internet
créer un fichier « ressources.qrc » avec le code suivant :

```
<!DOCTYPE RCC>
<RCC version="1.0">
<qresource>
<file alias="image1.ico">images/image1.ico</file>
<file alias="image2.ico">images/image2.ico</file>
</qresource>
</RCC>
```

pour créer le fichier python (mais dommage, ça ne marche pas bien en python !)
pyrcc4 -o qrc_resources.py resources.qrc

```
import qrc_resources
```

à rajouter en début de programme

Barre d'etat

➔ ajouter une barre d'etat, observer le résultat lors de la sélection des Actions

```
barreDetat = QStatusBar()   # barre d etat
barreDetat.showMessage("Pret")
self.setStatusBar(barreDetat)
```

rappel :

MDI (Multiple Document Interface) : elles peuvent afficher plusieurs documents à la fois. Elles affichent des sous-fenêtres dans la zone centrale. C'est le cas par exemple de Qt Designer

SDI (Single Document Interface) : elles ne peuvent afficher qu'un document à la fois. C'est le cas de Bloc-Notes, emacs

rajouter ces 3 lignes en début de code

```
self.zoneCentrale = QWorkspace()
self.setCentralWidget(self.zoneCentrale)
self.zoneCentrale.setScrollBarsEnabled(True)
```

dans les fonctions remplacer :

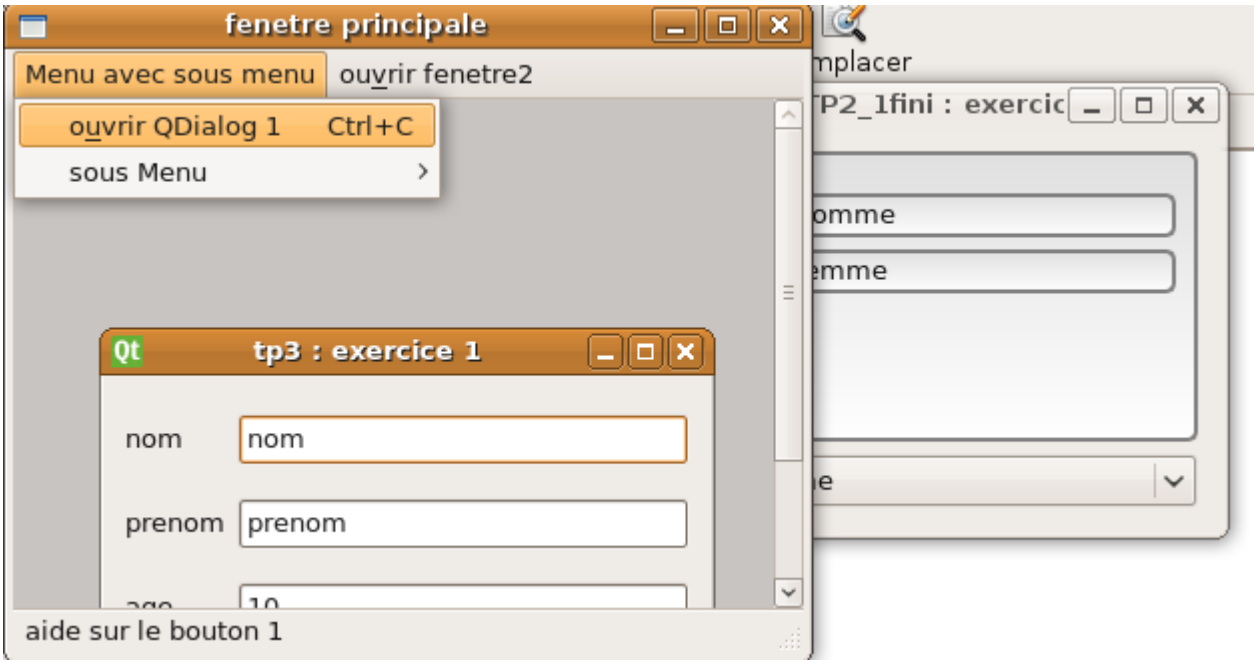
```
Fenetre.show()
Fenetre.exec_()
```

par

```
self.zoneCentrale.addWindow(Fenetre)
Fenetre.show()
```

➔ travail à réaliser : modifier votre code « exoTP3_3.py » avec les instructions ci-dessus pour ouvrir une fenêtre en mode MDI et une autre en mode SDI

obtenir la fenêtre suivante :



Barre d'outils

➔ réaliser une barre d'outils qui permet d'appeler les 2 actions précédentes

```
BarOutils = QToolBar ()
BarOutils.addAction(exoTP2_1fini_Action)
BarOutils.addAction(exoTP3_1fini_Action)
BarOutils.setAllowedAreas(Qt.TopToolBarArea | Qt.BottomToolBarArea) # autorise le positionnement en haut et en bas
self.addToolBar(BarOutils)
```

Dock Widgets

➔ réaliser un dock Widget (barre d'outils)

```
dock=QDockWidget("Palette",self)
self.addDockWidget(Qt.LeftDockWidgetArea, dock)
contenuDock = QWidget()
dock.setWidget(contenuDock)
```

```
label = QLabel("Exemple Dock :")
crayon = QPushButton("Crayon")
pinceau = QPushButton("Pinceau")
feutre =  QPushButton("Feutre")

dockLayout = QVBoxLayout()
dockLayout.addWidget(label)
dockLayout.addWidget(crayon)
dockLayout.addWidget(pinceau)
dockLayout.addWidget(feutre)
dockLayout.????????? A complèter : Facile
contenuDock.setLayout(dockLayout)
```

➔ travail à réaliser : modifier votre code « exoTP3_3.py » avec les instructions ci dessus pour obtenir la fenêtre suivante :

Les parties suivantes sont à titre d'information

4. Drag and Drop : présentation (copier ; coller)

Ergonomie :

Une définition du mot DRAG -AND-DROP, "Glisser-déposer" (ou Tirer-et-lâcher), dans une interface graphique, est le fait de faire glisser un objet sur un autre.

C'est quelque chose de difficile à programmer correctement : il faut

- identifier l'objet que l'on veut sélectionne (drag),
- identifier si le conteneur sur lequel on se déplace est prévu pour recevoir cet objet (move) et informer l'utilisateur (icone),
- recopier ou déplacer l'objet en mettant à jour le conteneur qui possédait l'objet et le conteneur qui reçoit l'objet, lors du lâcher de l'objet.

Cette partie reste complexe à comprendre et demande beaucoup de notions de programmation objet.

- Il est possible de rajouter des éléments pour permettre la complétion automatique.

```
items = QstringList()
items << self.tr("Matiere") << self.tr("cours") << self.tr("TD") <<
self.tr("TP") << self.tr("CODE")
self.completion_Auto=QCompleter(items)   # ne fonctionne pas a l'iut , dommage,
version encore trop ancienne, mais marche très bien en C++!
self.completion_Auto.setCaseSensitivity(Qt.CaseInsensitive)
self.LineEdit_nom.setCompleter(self.completion_Auto)
```

Ces différentes parties existent aussi en programmation Web