

26/09/2020

Python pour l'ingénieur



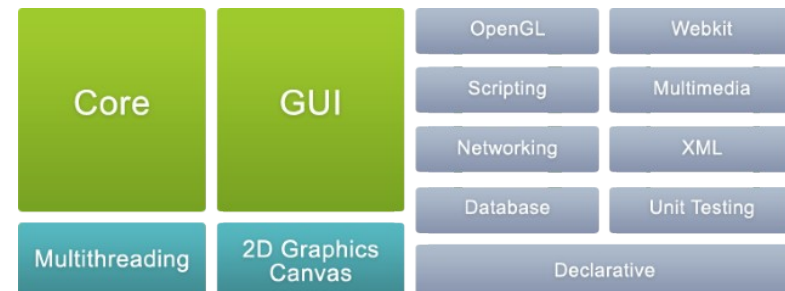
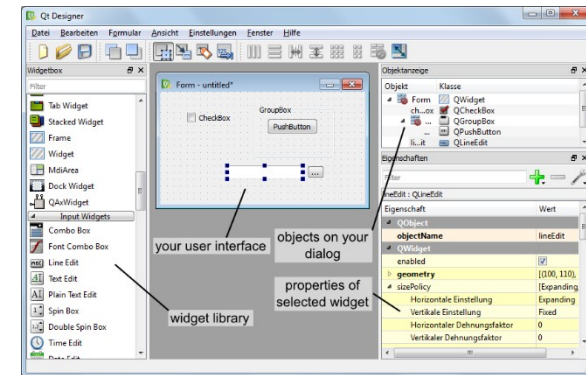
IHM avec PyQt

Module de formation

Introduction



- **Qt** est une bibliothèque de classes offrant entre autres des composants d'interface graphique appelés **widgets**.
- Qt est **multi-plateformes** (portable) et **open-source** (licence GNU LGPL permettant son utilisation légale et gratuite par des logiciels propriétaires).
- Qt est initialement écrit en langage C++.
- **PyQt** est un *binding* de Qt pour le langage Python.
 - PyQt n'est pas gratuit pour une utilisation commerciale
 - Autre binding Python de Qt : PySide
- Alternatives à Qt :
 - Python : Tk (intégré au langage), wxWidgets (wxPython)
 - C++ : Gtk, wxWidgets, MFC (Microsoft), etc.



Ressources

- Version de Qt utilisée pour le cours : **5.12** (6 Décembre 2018)
- Site internet de Pyqt :
 - <https://riverbankcomputing.com/software/pyqt/intro>
 - Documentation : <https://www.riverbankcomputing.com/static/Docs/PyQt5/sip-classes.html>
- Site internet de Qt :
 - <http://qt-project.org/>
 - Documentation : <https://doc.qt.io/qt-5/reference-overview.html>
- *La documentation de PyQt n'est pas aussi complète que celle de Qt → il faut parfois se référer à celle de Qt (en C++, mais la transcription en Python est assez facile).*

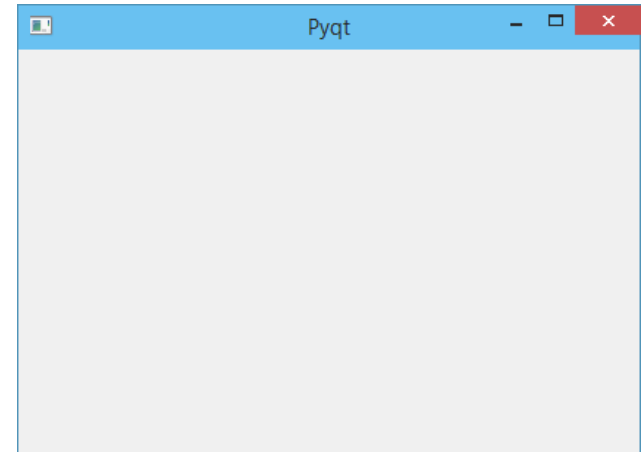
Application minimale

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

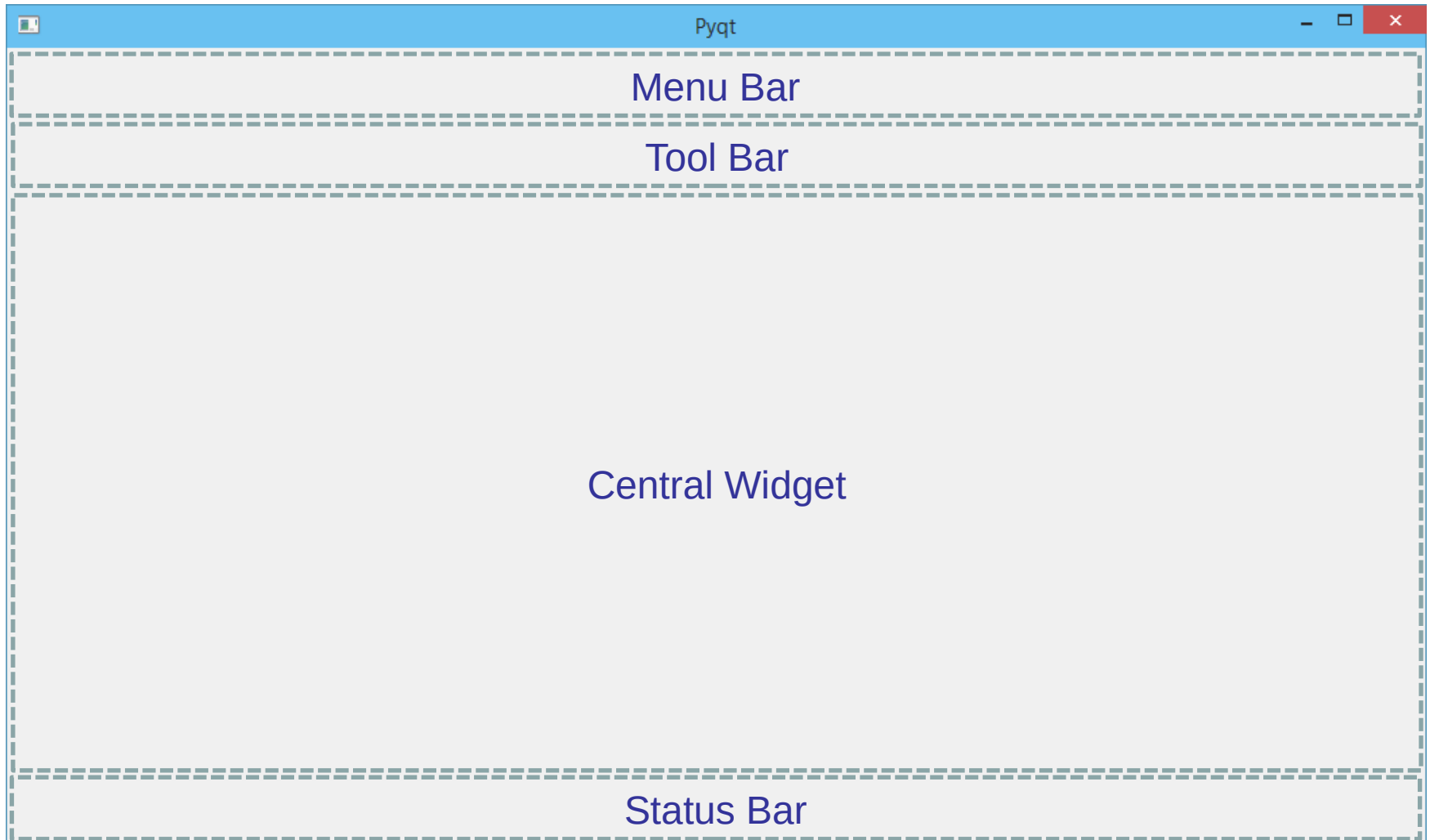
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')

def main():
    app = QApplication(sys.argv)
    fenetre = MaFenetre()
    fenetre.show()
    app.exec()

if __name__ == '__main__':
    main()
```



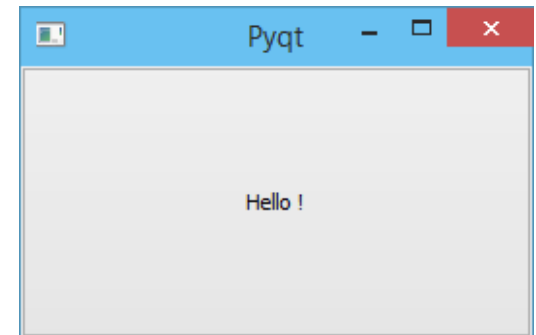
Layout de QMainWindow



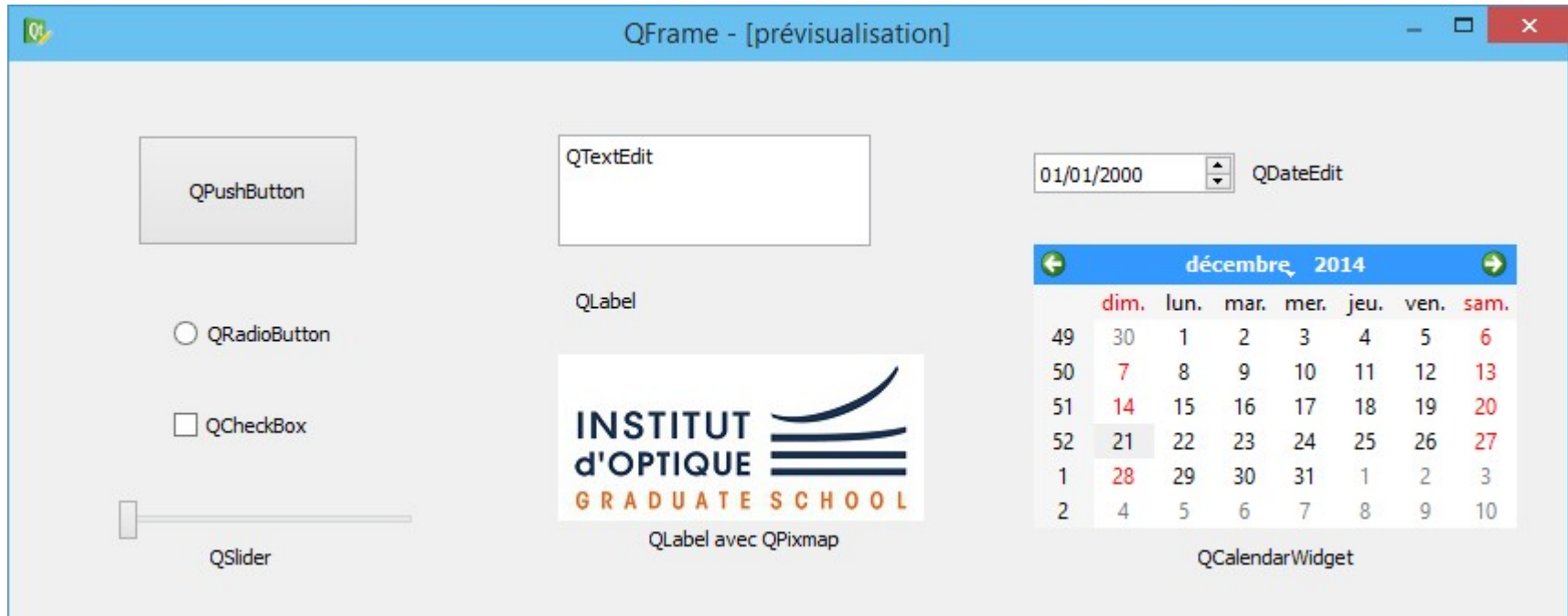
Central Widget

- Central Widget = un objet dérivant de QWidget
 - Soit un widget prédéfini
 - Soit un widget personnalisé, défini par une classe dérivant de QWidget

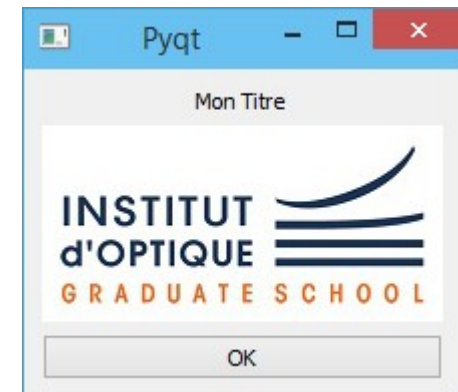
```
class MaFenetre(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle('Pyqt')  
  
        button = QPushButton('Hello !')  
  
        self.setCentralWidget(button)
```



Widgets prédéfinis

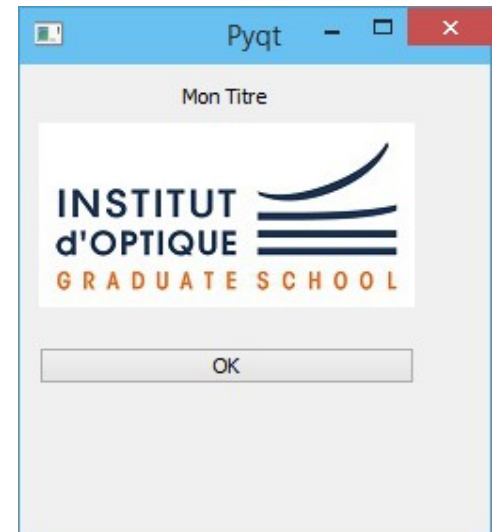


```
label = QLabel('Mon Titre', self)
label.setAlignment(Qt.AlignHCenter)
image = QLabel(self)
image.setPixmap(QPixmap('logo.jpg'))
bouton = QPushButton('OK', self)
```



Widget personnalisé

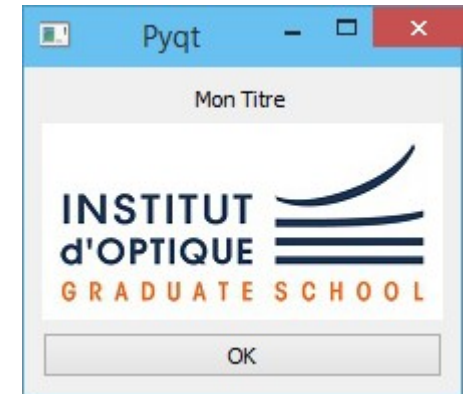
```
class MonWidget(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        label = QLabel('Mon Titre', self)  
        label.setAlignment(Qt.AlignHCenter)  
        label.setGeometry(10, 10, 200, 20)  
        image = QLabel(self)  
        image.setPixmap(QPixmap('logo.jpg'))  
        image.setGeometry(10, 30, 200, 100)  
        bouton = QPushButton('OK', self)  
        bouton.setGeometry(10, 150, 200, 20)  
  
class MaFenetre(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle('Pyqt')  
        central_widget = MonWidget(self)  
        self.setCentralWidget(central_widget)  
        self.resize(250, 250)
```



Placement avec Layout

```
class MonWidget(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        layout = QVBoxLayout()  
        label = QLabel('Mon Titre')  
        label.setAlignment(Qt.AlignHCenter)  
        image = QLabel()  
        image.setPixmap(QPixmap('logo.jpg'))  
        bouton = QPushButton('OK')  
        layout.addWidget(label)  
        layout.addWidget(image)  
        layout.addWidget(bouton)  
        self.setLayout(layout)
```

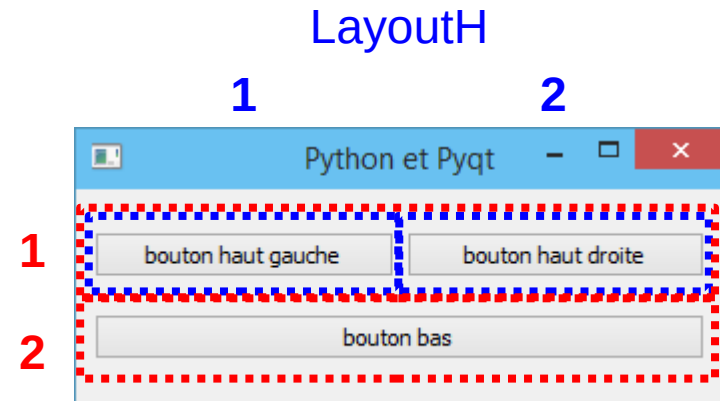
```
class MaFenetre(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle('Pyqt')  
        central_widget = MonWidget(self)  
        self.setCentralWidget(central_widget)
```



Imbrication des Layouts

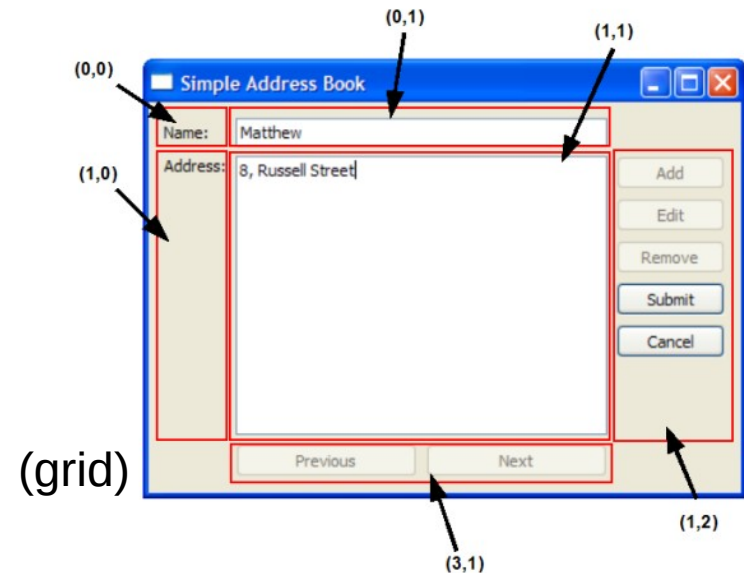
```
class MonWidget(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        bouton1 = QPushButton('bouton haut gauche')  
        bouton2 = QPushButton('bouton haut droite')  
        layoutH = QHBoxLayout()  
        layoutH.addWidget(bouton1)  
        layoutH.addWidget(bouton2)  
        bouton3 = QPushButton('bouton bas')  
        layoutV = QVBoxLayout()  
        layoutV.addLayout(layoutH)  
        layoutV.addWidget(bouton3)  
        self.setLayout(layoutV)
```

LayoutV



Types de Layout

- QHBoxLayout
- QVBoxLayout
- QGridLayout
- QFormLayout



Slot et Signal

```
class MonWidget(QWidget):
```

```
    def __init__(self):  
        super().__init__()
```

```
        self.bouton = QPushButton('Cliquez', self)  
        self.texte = QLineEdit(self)
```

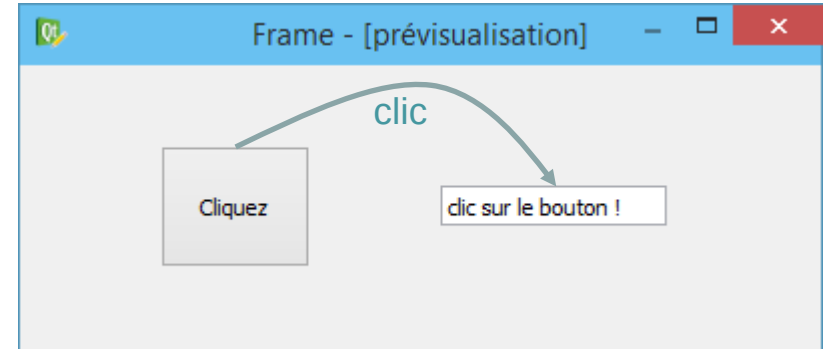
```
        self.bouton.clicked.connect(self.on_bouton)
```

```
        ...
```

↑
signal

↑
slot (ou 'callback')

```
    def on_bouton(self):  
        self.texte.setText('clic sur le bouton !')
```



Slot et Signal

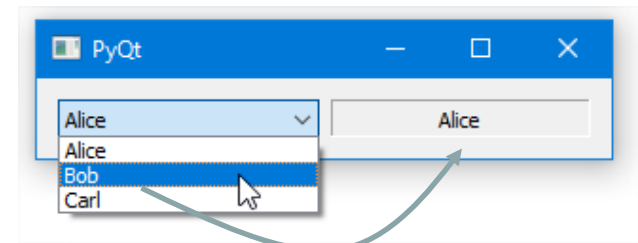
- Un signal peut être accompagné d'une information émise
Ex: `valueChanged(int)` pour `QSpinBox`, `QDial`
Ex: `currentTextChanged(str)` pour `QComboBox`
- Dans ce cas, la fonction callback doit avoir des arguments correspondant à ce qu'envoie le signal, pour réceptionner l'information émise :

```
class MonWidget(QWidget):  
    def __init__(self):  
        super().__init__()  
        self.label = QLabel()  
        combobox = QComboBox()  
        combobox.addItem("Alice")  
        combobox.addItem("Bob")  
        combobox.addItem("Carl")  
        combobox.currentTextChanged.connect(self.afficher_nom)
```

émet un str

reçoit un str

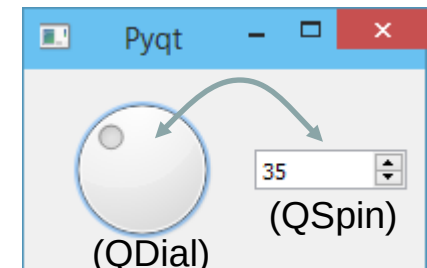
```
def afficher_nom(self, nom):  
    self.label.setText(nom)
```



```
dial = QDial(self)  
spin = QSpinBox(self)  
spin.valueChanged.connect(dial.setValue)  
dial.valueChanged.connect(spin.setValue)
```

émet un int

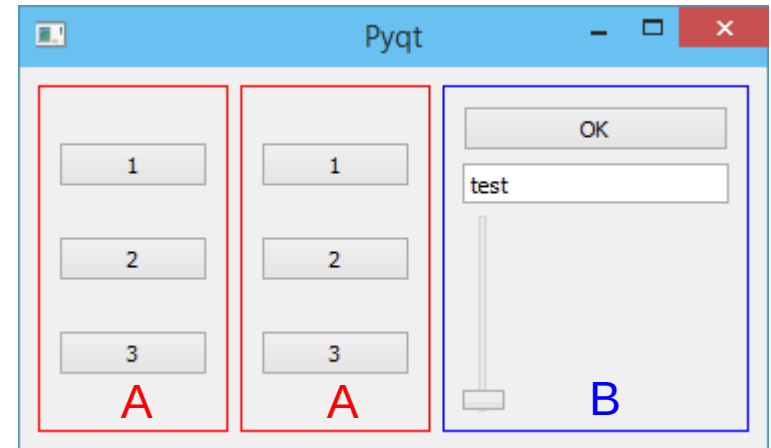
reçoit un int



Fenêtre plus complexe

```
class MonWidgetA(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        layout = QVBoxLayout()  
        layout.addWidget(QPushButton('1'))  
        layout.addWidget(QPushButton('2'))  
        layout.addWidget(QPushButton('3'))  
        self.setLayout(layout)
```

```
class MonWidgetB(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        layout = QVBoxLayout()  
        layout.addWidget(QPushButton('OK'))  
        layout.addWidget(QLineEdit('test'))  
        layout.addWidget(QSlider())  
        self.setLayout(layout)
```

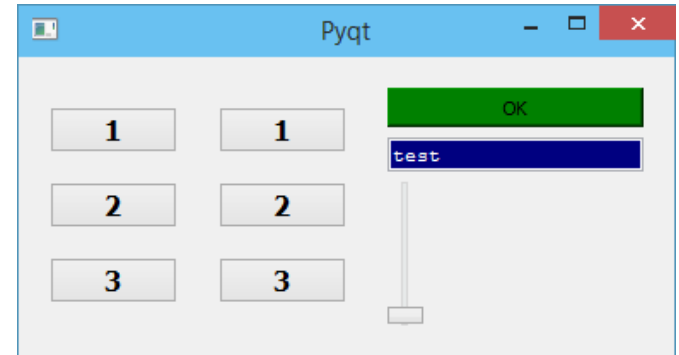


```
class MaFenetre(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        widget1 = MonWidgetA(self)  
        widget2 = MonWidgetA(self)  
        widget3 = MonWidgetB(self)  
        layout = QHBoxLayout()  
        layout.addWidget(widget1)  
        layout.addWidget(widget2)  
        layout.addWidget(widget3)  
        widget = QWidget()  
        widget.setLayout(layout)  
        self.setCentralWidget(widget)
```

Feuilles de style

```
class MonWidgetA(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        self.setStyleSheet('QPushButton { font-weight: bold; font-size: 16px; }')  
        layout = QVBoxLayout()  
        layout.addWidget(QPushButton('1'))  
        layout.addWidget(QPushButton('2'))  
        layout.addWidget(QPushButton('3'))  
        self.setLayout(layout)
```

```
class MonWidgetB(QWidget):  
    def __init__(self, parent):  
        super().__init__(parent)  
        self.setStyleSheet(' \n  
        QLineEdit { background-color: rgb(0,0,128); color: white; font-family: courier; } \n  
        QPushButton { background-color: rgb(0,128,0); }')  
        layout = QVBoxLayout()  
        layout.addWidget(QPushButton('OK'))  
        layout.addWidget(QLineEdit('test'))  
        layout.addWidget(QSlider())  
        self.setLayout(layout)
```



Menu déroulant

```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()

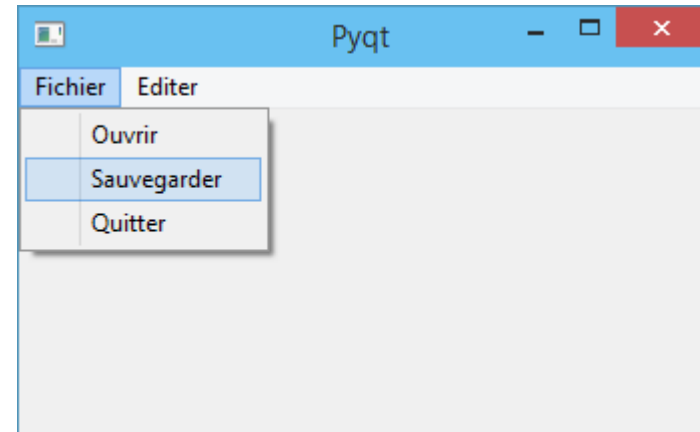
        menu1 = self.menuBar().addMenu('Fichier')

        action1 = QAction('Ouvrir', self)
        action1.triggered.connect(self.on_ouvrir)
        menu1.addAction(action1)

        action2 = QAction('Sauvegarder', self)
        action2.triggered.connect(self.on_sauver)
        menu1.addAction(action2)

        action3 = QAction('Quitter', self)
        action3.triggered.connect(self.on_quitter)
        menu1.addAction(action3)

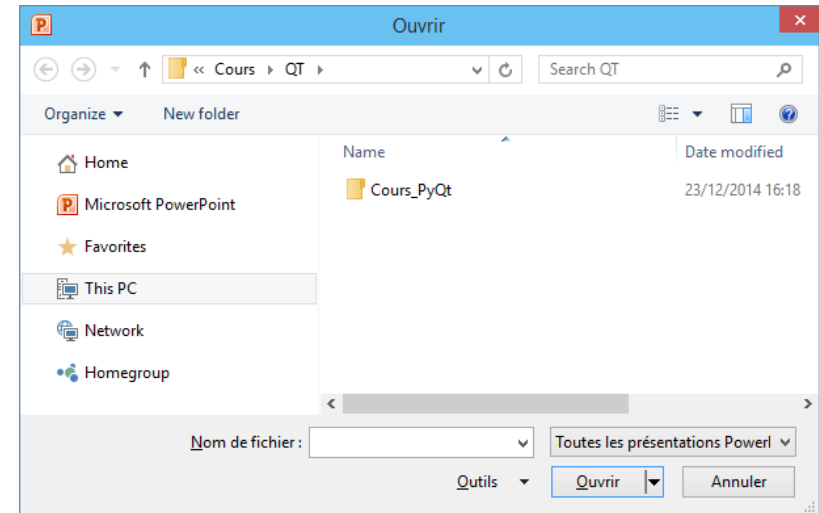
        menu2 = self.menuBar().addMenu('Editer')
```



↑ méthode de classe à définir

Quelques autres widgets de Qt

- Boîtes de dialogues usuelles
 - QMessageBox
 - QInputDialog
 - QFileDialog
- Widgets conteneurs
 - QGroupBox
 - QTabWidget
 - QStackedLayout
- Toolbar et Statusbar
 - QToolBar
 - QStatusBar
- Widgets complexes
 - QListView
 - QTreeView
 - QTableView

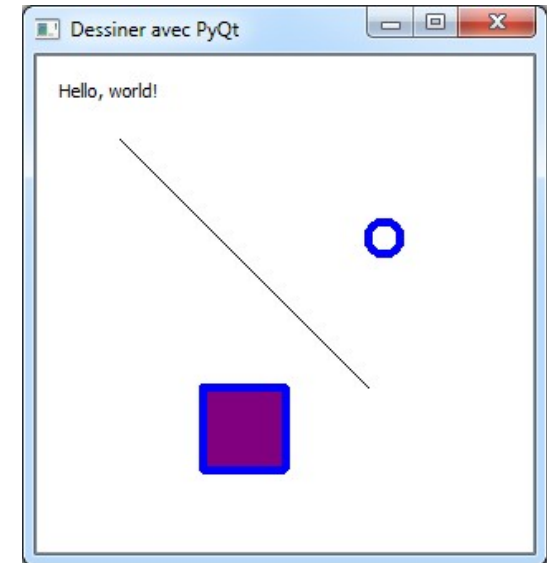


Dessiner

```
class MaScene(QGraphicsScene):
    """cette classe décrit la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        self.setSceneRect(0, 0, 300, 300)
        texte = self.addText("Hello, world!")
        texte.setPos(10, 10)
        self.addLine(50, 50, 200, 200)
        stylo = QPen(Qt.blue, 5, Qt.SolidLine)
        self.addEllipse(200, 100, 20, 20, stylo)
        brosse = QBrush(QColor(128, 0, 128), Qt.SolidPattern)
        self.addRect(100, 200, 50, 50, stylo, brosse)

class MaVueGraphique(QGraphicsView):
    """cette classe fait le rendu (= dessin) de la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        scene = MaScene(self)
        self.setScene(scene)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dessiner avec PyQt")
        vue = MaVueGraphique(self)
        self.setCentralWidget(vue)
```



Clavier et Souris

- Toute classe dérivée de QWidget peut redéfinir des fonctions héritées pour réagir aux événements clavier et souris, par exemple :

- `keyPressEvent()`

- appelée lorsque qu'une touche du clavier est pressée.

```
def keyPressEvent(self, keyevent):  
    if keyevent.key() == Qt.Key_Q:  
        ...
```

- `mousePressEvent()`

- appelée lorsqu'un bouton de la souris est cliqué.

```
def mousePressEvent(self, mouseevent):  
    self.x = mouseevent.scenePos().x()  
    self.y = mouseevent.scenePos().y()
```

Timer

- Un *timer* permet de déclencher l'appel d'une fonction à intervalles de temps réguliers.
- La classe Qt pour créer un timer est **QTimer**

```
def afficher():  
    print("bonjour")  
  
app = QApplication(sys.argv)  
timer = QTimer()  
timer.timeout.connect(afficher)  
# répétition toutes les 1000 millisecondes  
timer.start(1000)  
app.exec()
```

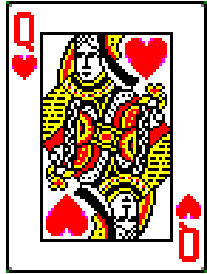
```
class MonTimer(QTimer):  
    def __init__(self):  
        super().__init__()  
        self.timeout.connect(self.ontimer)  
  
    def ontimer(self):  
        print("bonjour")  
  
app = QApplication(sys.argv)  
timer = MonTimer()  
# répétition toutes les 1000 millisecondes  
timer.start(1000)  
app.exec()
```

`timer.stop()` permet d'arrêter le timer

Exercice

- 1^{re} partie

- Créer une classe (scène) affichant une image de carte.
`addPixmap(QPixmap("carte.png"))`
- La classe contient une fonction permettant de repositionner cette carte à l'endroit où on clique.
- La classe contient une fonction permettant de créer une 2^e carte quand on appuie sur la touche 'c'.
- La classe contient une fonction permettant de déplacer la 2^e carte quand on appuie sur les flèches du clavier.

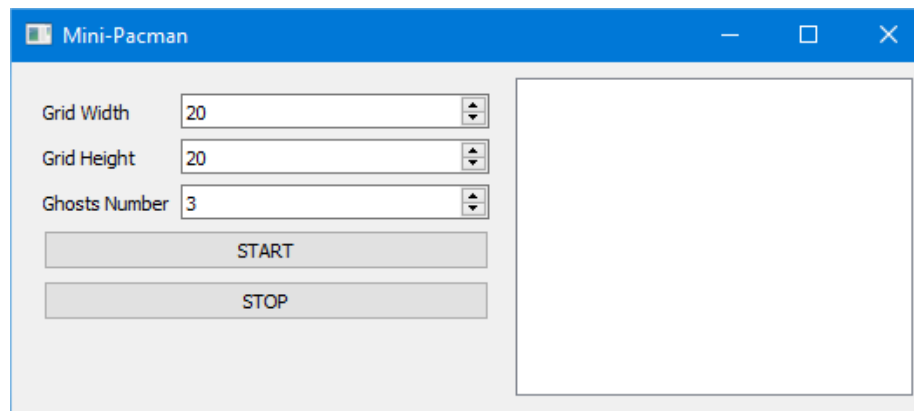


- 2^e partie

- La classe contient un timer permettant d'animer la première carte, en modifiant sa position de quelques pixels à intervalles de temps réguliers.
- La classe contient une fonction permettant de mettre l'animation en pause quand on appuie sur la touche 'p'.

Exercice : mini Pacman (3^e partie)

- Partir des modules `model.py`, `controler.py` et `view.py` fournis sur learnpython.ovh
- Module `view.py`:
 - coder le constructeur de la classe `PacmanParams` de manière à obtenir l'interface ci-dessous
 - Utiliser `QSpinBox` pour saisir les dimensions du plateau de jeu et le nombre de fantômes
 - Utiliser `QFormLayout` pour titrer et disposer les 3 options
 - connecter les méthodes `on_start` et `on_stop` aux boutons `start` et `stop`. Ces méthodes ne font rien pour le moment.



Exercice : mini Pacman (3^e partie)

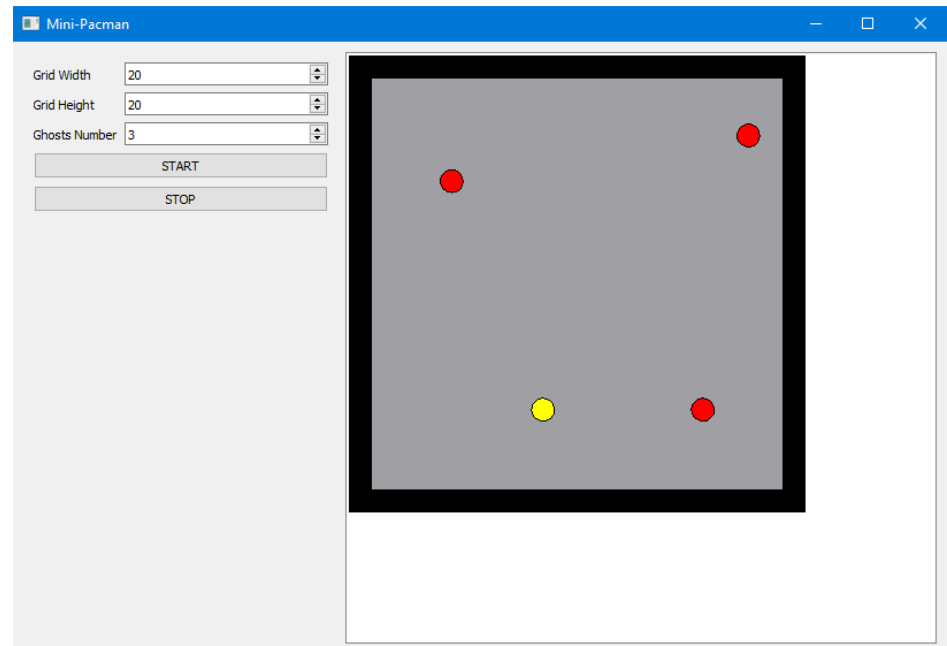
- Module `contrôler.py`
 - Coder le constructeur et la méthode `start()` de `PacmanControler`
 - Attributs de la classe `PacmanControler` :
 - `self.ghosts` : liste d'objets fantômes (`Entity`)
 - `self.pacman` : objet pacman (`Pacman`)
 - `self.timer` : timer
 - `self.width` : largeur du plateau
 - `self.height` : hauteur du plateau
 - Dans le constructeur :
 - Créer l'attribut timer (classe `QTimer`) et le connecter à la méthode `next()`
 - Dans `start()`:
 - Créer `nb_ghosts` objets fantômes et leur donner une position initiale aléatoire
 - Créer un objet pacman et lui donner une position initiale aléatoire
 - Démarrer le timer
 - Dans `stop()`
 - Arrêter le timer
 - Appeler les méthodes `start()` et `stop()` du contrôleur au moment adéquat dans la classe `PacmanParams` (`view.py`). Bien prendre en compte les paramètres saisis dans l'interface (largeur, hauteur, nombre de fantômes). Vérifier que ces paramètres parviennent avec la bonne valeur à la méthode `start()`.
 - Vérifier que la méthode `next()` est bien appelée à intervalle de temps régulier lorsqu'on clique sur le bouton Start (mettre un print dans la méthode `next()`). Vérifier que cliquer sur le bouton Stop arrête bien le timer.

Exercice : mini Pacman (3^e partie)

- Module view.py:
 - Classe PacmanScene :
 - Constructeur :
 - Initialiser la scène à une dimension de 512 x 512 pixels
 - Méthode refresh() :
 - Dessiner le plateau de jeu en allouant 25x25 pixels à chaque case du plateau de jeu
 - » Dessiner le fond en gris et les contours du plateau en noir (rectangles)
 - » Dessiner chaque fantôme (disque de couleur rouge)
 - » Dessiner Pacman (disque de couleur jaune)

- Module controler.py
 - Compléter la méthode next() :
 - Déplacer les fantômes
 - Déplacer Pacman
 - Si un fantôme est "mangé", le supprimer de la liste des fantômes
 - S'il n'y a plus de fantôme, arrêter le timer
 - Provoquer un rafraichissement de tous les "clients" du contrôleur

- Tester
 - Vérifier le bon déplacement des entités
 - Vérifier la disparition des fantômes mangés
 - Vérifier que la fin de partie se passe bien (Pacman s'arrête)
 - Vérifier qu'on peut démarrer une nouvelle partie, en modifiant ou pas les paramètres du jeu



Exercice : mini Pacman (3^e partie)

- Module view.py:
 - Téléportation de Pacman à la souris :
 - Ajouter une méthode `move_pacman(x, y)` au contrôleur qui téléporte instantanément Pacman à la position (x, y). Ne pas permettre de téléporter Pacman en dehors du plateau de jeu ou sur la bordure.
 - Coder la méthode `mousePressEvent` pour que Pacman soit téléporté sur le curseur de souris lorsqu'on clique sur le plateau de jeu.
 - Mettre le jeu en pause au clavier (touche P)
 - Implémenter la méthode `keyPressEvent()` (module view.py) pour transmettre la touche du clavier saisie à la méthode `process_keypress()` du contrôleur
 - Implémenter la méthode `process_keypress()` (module controler.py) pour activer / désactiver la pause par la touche P. La mise en pause consiste à stopper le timer. Redémarrer le timer pour enlever la pause, sans réinitialiser le jeu.
 - Vérifier le bon comportement si le jeu est relancé (bouton start) pendant la pause.

Exercice : mini Pacman (3^e partie)

- Quelques améliorations :
 - Adapter automatiquement la dimension en pixels d'une case de plateau (actuellement fixé à 25x25 pixels) pour que la plateau occupent toujours 512 pixels quel que soit la largeur et la hauteur choisies
 - Lorsqu'il n'y a plus de fantômes, afficher "Game Over" par-dessus le plateau de jeu (jusqu'au démarrage d'une nouvelle partie avec start)
 - Afficher le nombre de fantômes encore en vie dans la barre des paramètres
 - Ajouter un paramètre permettant de modifier la vitesse du jeu pendant une partie (ajouter un widget QSpinBox dans les paramètres, utiliser la méthode setInterval du timer).

