LECTURE 17

GUI Programming

GUI PROGRAMMING

Today, we're going to begin looking at how we can create GUIs (Graphical User Interfaces) in Python.

So far, every application we've built has been either console-based or a web application. If we want to create a user-friendly standalone application, we really must create a nice interface for it.

As an example, we'll create an interface for the triangle peg game.

First, let's see what packages are available to help us do this.

GUI PROGRAMMING IN PYTHON

There are a huge number of modules available to help you create an interface. Some of these include:

- Tkinter: wrapper around Tcl/Tk. Python's standard GUI.
- PyQt: bindings for the Qt application framework.
- wxPython: wrapper around wxWidgets C++ library.
- pyGTK: wrapper around GTK+.
- PyJamas/PyJamas Desktop
- etc.

CHOOSING A TOOLKIT

Toolkit	Important Points
Tkinter	- Limited theme support: “look” was relatively the same until recently. - Relatively limited widget options. - Bundled with Python since the beginning. - Most commonly used.
PyQt	- Limited licensing choices. - Very good docs - Very beautiful applications. - Huge! Both good and bad...
wxPython	- Very large user base. - Great widget selection. - Bad docs. - Not Python 3 compatible.
pyGTK	- Originally developed for GIMP. - Stable with a full selection of widgets. - Only offers theme-able native widgets on Windows + Linux. Mac lags behind somewhat. - Some quirks to work around (some C programming style).

PYQT

In this class, we will be exploring basic GUI development with PyQt since it is a widely-used toolkit. However, the other mentioned options are all very good and it may be necessary to branch out depending on the complexity and/or needs of the application.

First, we'll discuss some of the mechanics of GUI development. Afterwards, we could spend tons of class time learning about PyQt's ~1000 classes but we won't. What we'll do is build an application together and get familiar with the common parts of a PyQt application.

As a note, we'll be using PyQt5 in this lecture. A PyQt4 tutorial is included in the old lecture but the details aren't very different.

GETTING PYQT5

1. Go to https://www.riverbankcomputing.com/software/sip/download and grab the latest version of SIP.

2. Run the following commands to build and install SIP (version numbers may be different).

```
$ gunzip sip-4.18.tar.gz
$ tar -xvf sip-4.18.tar
$ cd sip-4.18/
$ python configure.py
$ make
$ sudo make install
```

GETTING PYQT5

1. Now go to https://www.riverbankcomputing.com/software/pyqt/download5 and grab the latest version of PyQt5.

2. Run the following commands to build and install PyQt5 (version numbers may be different).

```
$ gunzip PyQt5_gpl-5.6.tar.gz
$ tar -xvf PyQt5_gpl-5.6.tar
$ sudo apt-get install qt5-default
$ cd PyQt5_gpl-5.6
$ python configure.py
$ make          ← go get some coffee, this one will take a while!
$ sudo make install
```

3. Open up an interpreter and `import PyQt5` to make sure everything's working.

GUI PROGRAMMING IN PYQT

PyQt is a multi-platform GUI toolkit. It has approximately ~1000 classes divided into a set of ~38 modules. Among these are QtCore and QtGui – the most commonly used PyQt modules.

- **QtCore** contains the core classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for animations, state machines, threads, mapped files, shared memory, regular expressions, and user and application settings.
- **QtGui** contains classes for windowing system integration, event handling, 2D graphics, basic imaging, fonts and text.
- **QtWidgets** contains classes that provide a set of UI elements to create classic desktop-style user interfaces.

Besides these, there are other modules (like, QtNetwork and QtOpenGL) which focus on specific functionality.

GUI PROGRAMMING IN PYQT

Programming a GUI is not that different than programming an object-oriented console application.

What is different is that GUI programming involves the use of a *Toolkit* and GUI developers must follow the pattern of program design specified by the toolkit.

As a developer, you should be familiar with the API and design rules of at least one toolkit – preferably one that is multiplatform with bindings in many languages!

GUI PROGRAMMING IN PYQT

GUI programming necessarily means **object-oriented programming** with an **event-driven framework**. This should make sense: your job as a GUI developer is to create an application that responds to events.

For instance, when a user clicks their mouse on a certain button, the program should do X. When the user presses enter in a text field, the program should do Y. You're defining the behavior of the program as a response to external events.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = QtWidgets.QWidget()
    main_window.show()
    app.exec_()
```

The QApplication class manages the application's control flow and main settings. It controls the main event loop through which all events are handled and scheduled. No matter how many windows there are, there is only one QApplication instance.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = QtWidgets.QWidget()
    main_window.show()
    app.exec_()
```

A **widget** is a control element which is visible and can be manipulated by the user. These include elements such as buttons, text fields, radio selections, etc.

If we create a basic widget QWidget instance without a parent widget, it automatically becomes a window.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = QtWidgets.QWidget()
    main_window.show()
    app.exec_()
```

QWidget implements a variety of methods for it and its derived classes. These include resize, move, setWindowTitle (for widgets with no parents), among many others.

Widgets and their children are created in memory and made visible with the show() method.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

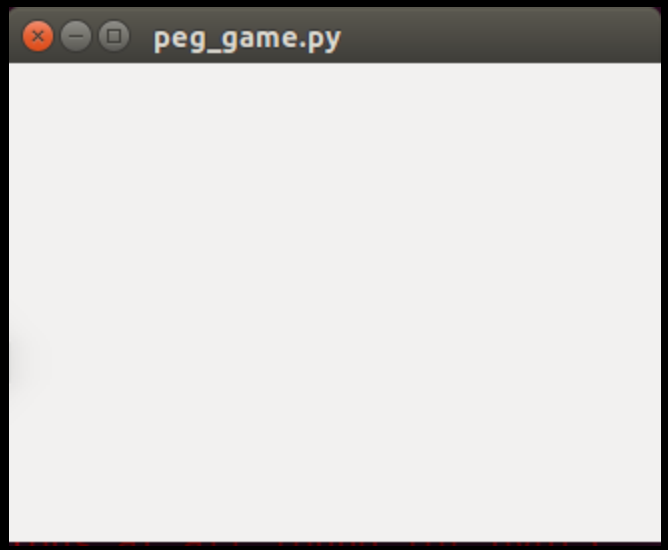
```
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = QtWidgets.QWidget()
    main_window.show()
    app.exec_()
```

Calling exec_() on our QApplication instance will start our main event loop.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

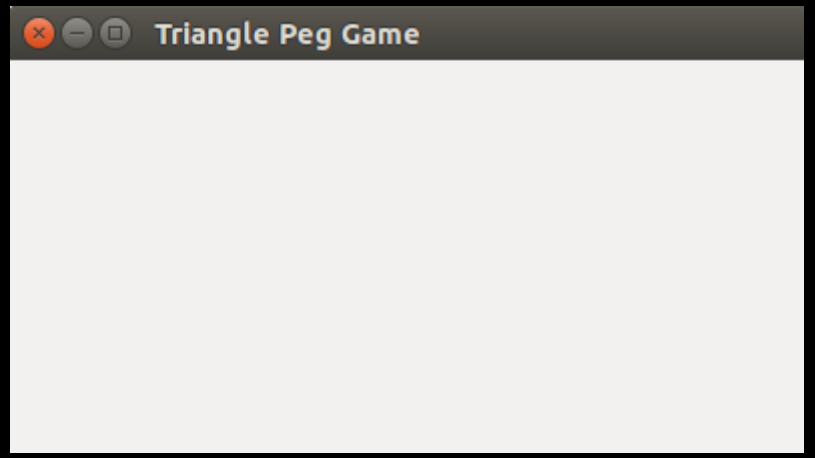
```
from PyQt5 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = QtWidgets.QWidget()
    main_window.show()
    app.exec_()
```



BASIC PYQT

Rather than the procedural approach we took before, we should try to define our interface in an object-oriented manner.

```
from PyQt5 import QtWidgets

class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setGeometry(200, 200, 400, 200)
        self.setWindowTitle('Triangle Peg Game')
        self.show()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = PegGameWindow()
    app.exec_()
```



QWIDGET

So, we can see that a parentless QWidget instance gives us a window, but the QWidget class is actually the base class for all UI elements. Some of the classes that inherit and extend QWidget include:

- QProgressBar
- QPushButton
- QCheckBox
- QScrollBar
- and many, *many* more!

QWIDGET

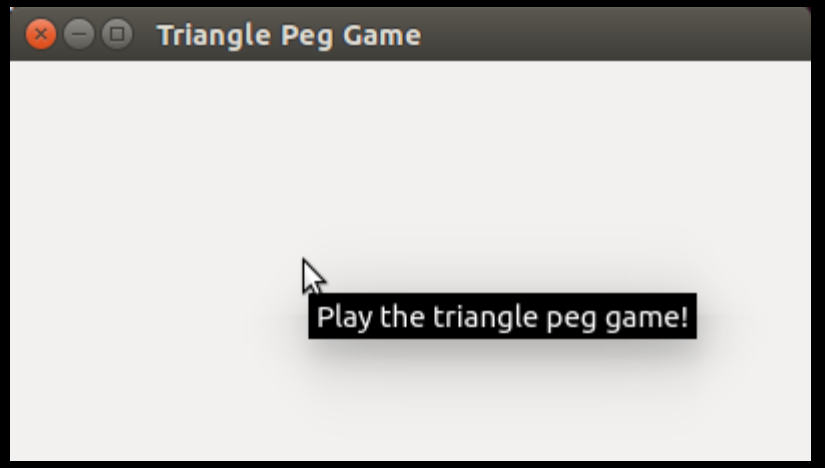
The QWidget class also defines some of the basic functionality common to all widgets.

- `QWidget.geometry()` **and** `Qwidget.setgeometry(`*`x, y, w, h`*`)`
- `QWidget.resize(`*`w, h`*`)`
- `QWidget.setParent(`*`parent`*`)`
- `QWidget.setToolTip(`*`str`*`), QWidget.setStatusTip(`*`str`*`)`
- `QWidget.setPalette(`*`palette`*`)`

Check here for an exhaustive list. Not only is this not all of the methods defined – we're not even including all the different *ways* you can call these methods!

BASIC PYQT

```
class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setGeometry(200, 200, 400, 200)
        self.setWindowTitle('Triangle Peg Game')
        self.setToolTip("Play the triangle peg game!")
        self.show()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = PegGameWindow()
    app.exec_()
```



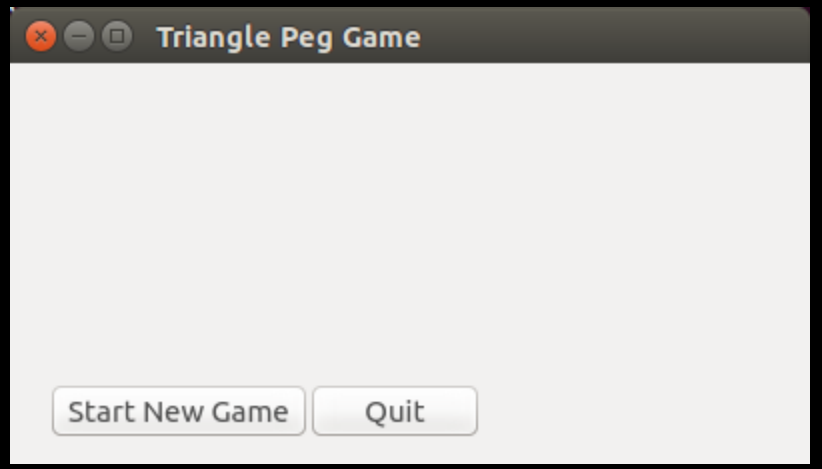
BASIC PYQT

Let's make some buttons!

```
Class StartNewGameBtn(QtWidgets.QPushButton):
    def __init__(self, parent):
        QtWidgets.QPushButton.__init__(self, parent)
        self.setText("Start New Game")
        self.move(20,160)

class QuitBtn(QtWidgets.QPushButton):
    def __init__(self, parent):
        QtWidgets.QPushButton.__init__(self, parent)
        self.setText("Quit")
        self.move(150,160)
```

BASIC PYQT

```
class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setup()
    def setup(self):
        self.setGeometry(200, 200, 400, 200)
        self.setWindowTitle('Triangle Peg Game')
        self.setToolTip("Play the triangle peg game!")
        self.new_button = StartNewGameBtn(self)
        self.quit_button = QuitBtn(self)
        self.show()
```



BASIC PYQT

Alternatively….

```
class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.setup()

    def setup(self):
        self.setGeometry(200, 200, 400, 200)
        self.setWindowTitle('Triangle Peg Game')
        self.setToolTip("Play the triangle peg game!")
        self.new_button = QtWidgets.QPushButton("Start New Game", self)
        self.new_button.move(20, 160)
        self.quit_button = QtWidgets.QPushButton("Quit", self)
        self.quit_button.move(150, 160)
        self.show()
```

SIGNALS AND SLOTS

PyQt5 makes use of a signal/slot mechanism for specifying the actions that should be taken when an event happens.

A *signal* is emitted when a particular event occurs. Widgets already have many predefined signals, but you could also create custom signals on subclassed widgets.

Some common signals of `QPushButton` are:

- `QPushButton.clicked` – signal activated when button is pressed and then released while mouse is on button.
- `QPushButton.pressed` – signal activated when button is pressed.
- `QPushButton.released` – signal activated when button is released.

SIGNALS AND SLOTS

The example buttons we created earlier can be clicked, but nothing will happen. That's because we need to assign a *slot* to the signal.

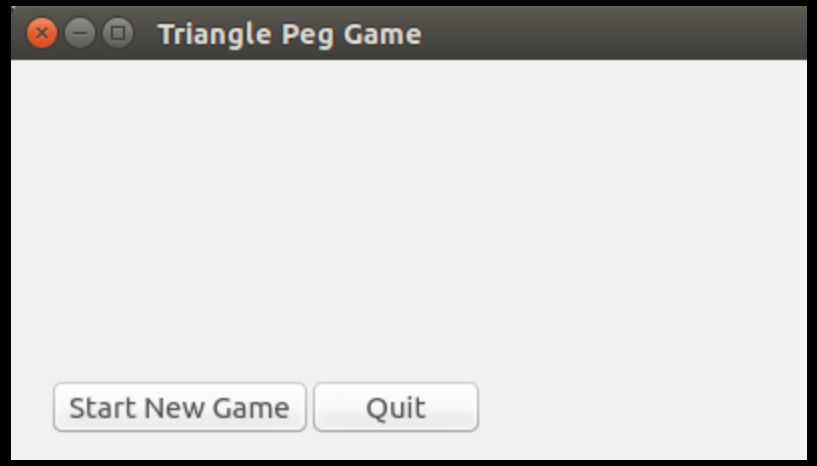
A slot is a function that is called in response to a particular signal. Widgets have many pre-defined slots, but because a slot can be any Python callable, we can easily define our own slots to define our reaction to the signal.

Let's try to make our Quit button exit the application. The slot we want to assign is:

```
QWidgets.qApp.quit()
```

Note that `QWidgets.qApp` is the `QApplication` instance running our program.

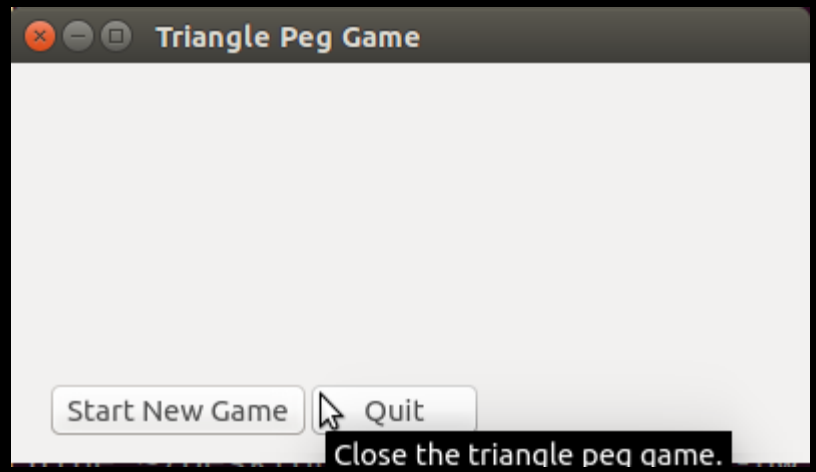
SIGNALS AND SLOTS




```
class QuitBtn(QtWidgets.QPushButton):
    def __init__(self, parent):
        QtWidgets.QPushButton.__init__(self, parent)
        self.setText("Quit")
        self.move(150,160)
        self.clicked.connect(QtWidgets.qApp.quit)
        self.setToolTip("Close the triangle peg game.")
```

Clicking quit now causes our window to close!

SIGNALS AND SLOTS




```
class QuitBtn(QtWidgets.QPushButton):
    def __init__(self, parent):
        QtWidgets.QPushButton.__init__(self, parent)
        self.setText("Quit")
        self.move(150,160)
        self.clicked.connect(QtWidgets.qApp.quit)
        self.setToolTip("Close the triangle peg game.")
```

Clicking quit now causes our window to close!

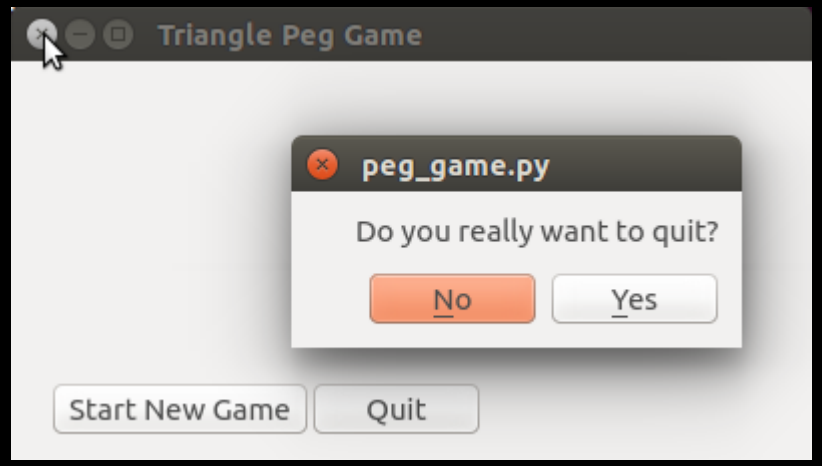
BASIC PYQT

We are also free to define the behavior of our application by overriding built-in methods.

For example, `QtWidgets.QWidget` has a method called `closeEvent()` which is called automatically with an instance of a window close request. By default, we just accept the request and close the window. Here, we'll override the function to ask if they're sure.

```
class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        ...
    def setup(self):
        ...
    def closeEvent(self, event):
        reply = QuitMessage().exec_()
        if reply == QtWidgets.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()


class QuitMessage(QtWidgets.QMessageBox):
    def __init__(self):
        QtWidgets.QMessageBox.__init__(self)
        self.setText("Do you really want to quit?")
        self.addButton(self.No)
        self.addButton(self.Yes)
```

BASIC PYQT



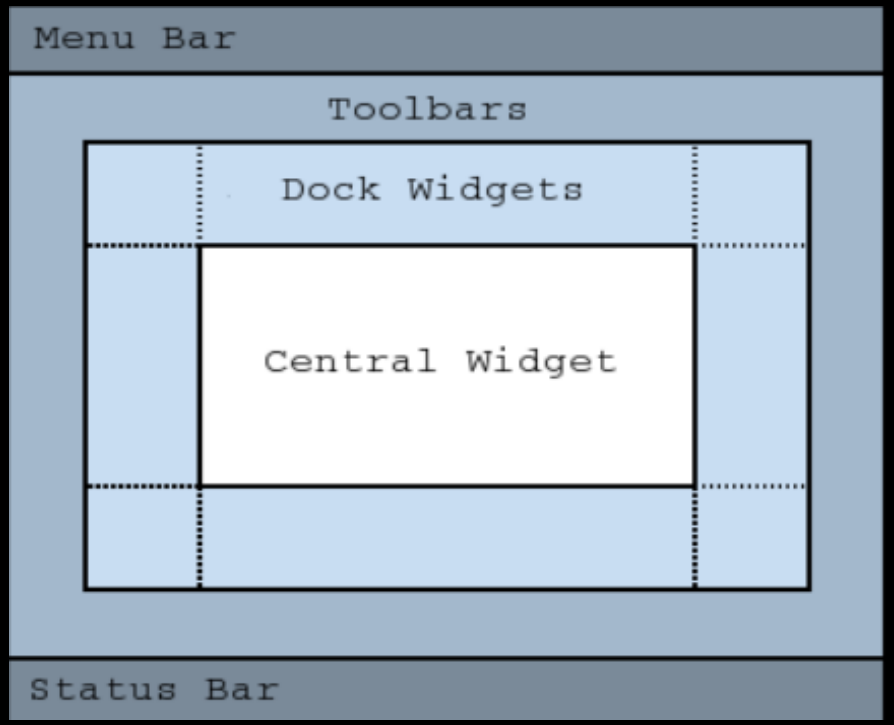

```
class PegGameWindow(QtWidgets.QWidget):
    def __init__(self):
        ...
    def setup(self):
        ...
    def closeEvent(self, event):
        reply = QuitMessage().exec_()
        if reply == QtWidgets.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()


class QuitMessage(QtWidgets.QMessageBox):
    def __init__(self):
        QtWidgets.QMessageBox.__init__(self)
        self.setText("Do you really want to quit?")
        self.addButton(self.No)
        self.addButton(self.Yes)
```

QT MAIN WINDOW

The `QtWidgets.QMainWindow` class provides a main application window. `QMainWindow` has its own layout as opposed to `QWidget` (but it inherits from `QWidget`).

`QMainWindow` automatically makes room for:

- `QToolBar`
- `QDockWidget`
- `QMenuBar`
- `QStatusBar`
- Any widget can occupy Central Widget.



BASIC PYQT

We're going to add a menu bar with a File > Exit action.

First, we create a `QAction` instance for exiting the application, for which the text is "Exit".

When the action is taken, we quit the application.

```
class PegGameWindow(QtWidgets.QMainWindow):
    def __init__(self):
        ...
    def setup(self):
        ...
        self.central_widget = QtWidgets.QWidget(self)
        self.new_button = StartNewGameBtn(self.central_widget)
        self.quit_button = QuitBtn(self.central_widget)
        self.setCentralWidget(self.central_widget)

        exit_action = QtWidgets.QAction('Exit', self)
        exit_action.triggered.connect(QtWidgets.qApp.quit)

        menu_bar = self.menuBar()
        file_menu = menu_bar.addMenu('File')
        file_menu.addAction(exit_action)

        self.show()
```

BASIC PYQT

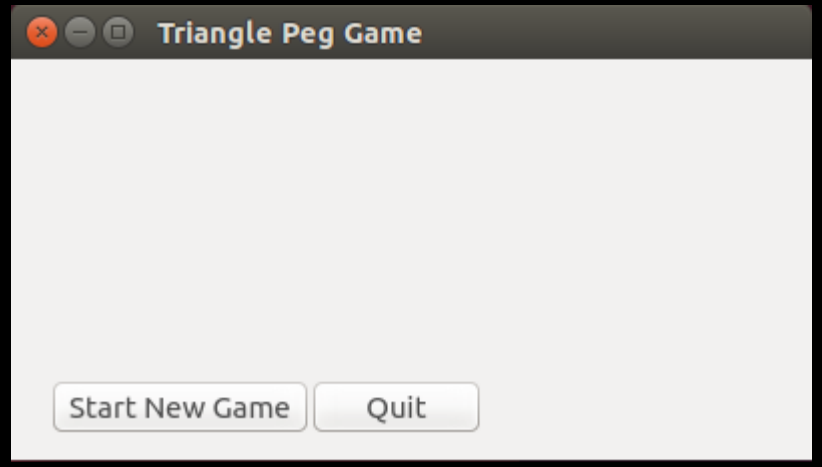
We reference the menu bar for our main window by calling menuBar(). This will also create and return an empty menu bar if there is none.

Then we add a new menu with the text “File”. Then we add the Exit action to the File menu.

```
class PegGameWindow(QtWidgets.QMainWindow):
    def __init__(self):
        ...
    def setup(self):
        ...
        self.central_widget = QtWidgets.QWidget(self)
        new_button = StartNewGameBtn(self.central_widget)
        quit_button = QuitBtn(self.central_widget)
        self.setCentralWidget(self.central_widget)

        exit_action = QtWidgets.QAction('Exit', self)
        exit_action.triggered.connect(QtWidgets.qApp.quit)

        menu_bar = self.menuBar()
        file_menu = menu_bar.addMenu('File')
        file_menu.addAction(exit_action)

        self.show()
```

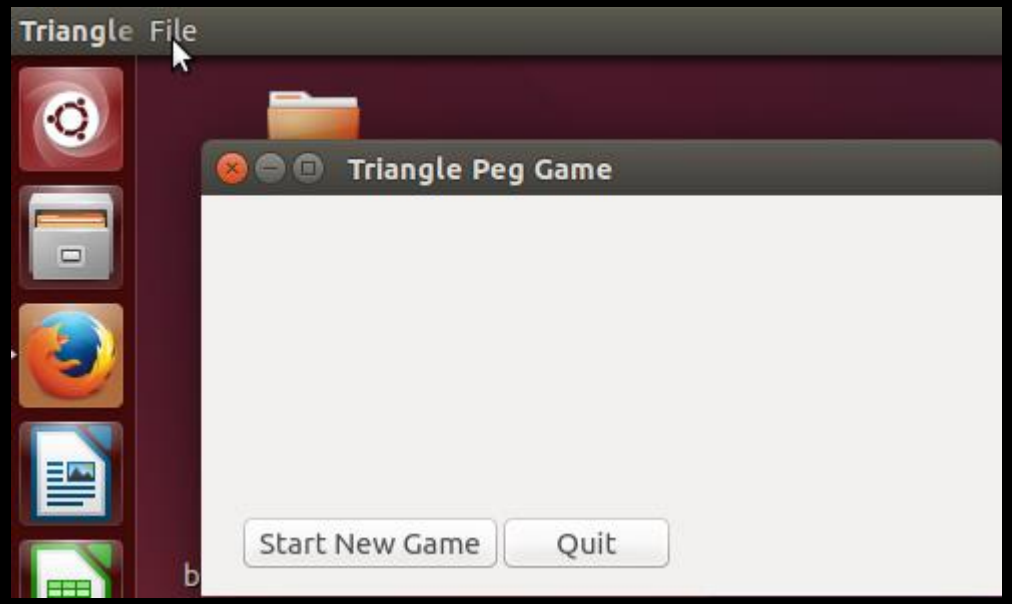
BASIC PYQT

Where's our menu bar!?



BASIC PYQT

Where's our menu bar!?



Very sneaky, Ubuntu…

BASIC PYQT

```
menu_bar = self.menuBar()
menu_bar.setNativeMenuBar(False)
file_menu = menu_bar.addMenu('File')
file_menu.addAction(exit_action)
```