1. GUI Programmierung

1.1. Einführung

Python besitzt eine Vielzahl von GUI-Frameworks (bzw. Toolkits). Es gibt plattformunabhängige, aber auch plattformspezifische Technologien[1].

Die wichtigsten GUI-Frameworks basieren auf Gtk, Qt, Tk und WxWidgets.

1.2. Qt

Mittels Qt5 kann ein GUI inklusive der notwendigen Funktionalität sehr einfach erstellt werden. Neben PyQt5 ist auch das Community Projekt PySide[2] eine Umsetzung für das Qt cross-platform GUI/XML/SQL C++ framework[3].
Während PySide die Version Qt 4.8 verwendet, ist bei PyQt sowohl Qt4 als auch Qt5 verfügbar.
PyQt[4] wird von der Firma Riverbank Computing Limited entwickelt.

Die Software kann von Sourceforge geladen werden:
http://sourceforge.net/projects/pyqt/files/PyQt5/

Die aktuelle Version für Python 3.3 ist PyQt 5.2.1 und PySide 1.24.

1.2.1. Installation

Die Installation erfolgt abhängig von der verwendeten Plattform in das Lib/site-packages/PyQt5 bzw. Lib/site-packages/PySide Verzeichnis.
Für die Erstellung eines GUIs wird der „designer“ verwendet, welcher sich nach der Installation ebenfalls in diesem Verzeichnis befindet.

1.2.2. Erstellung eines GUIs

Mithilfe des „designer“-Programmes kann sehr einfach ein neues grafisches Benutzerinterface erstellt werden:

[1] Siehe https://wiki.python.org/moin/GuiProgramming
[2] Siehe qt-project.org/wiki/PySide
[3] Siehe https://wiki.python.org/moin/PyQt
[4] Siehe http://www.riverbankcomputing.com/software/pyqt/intro

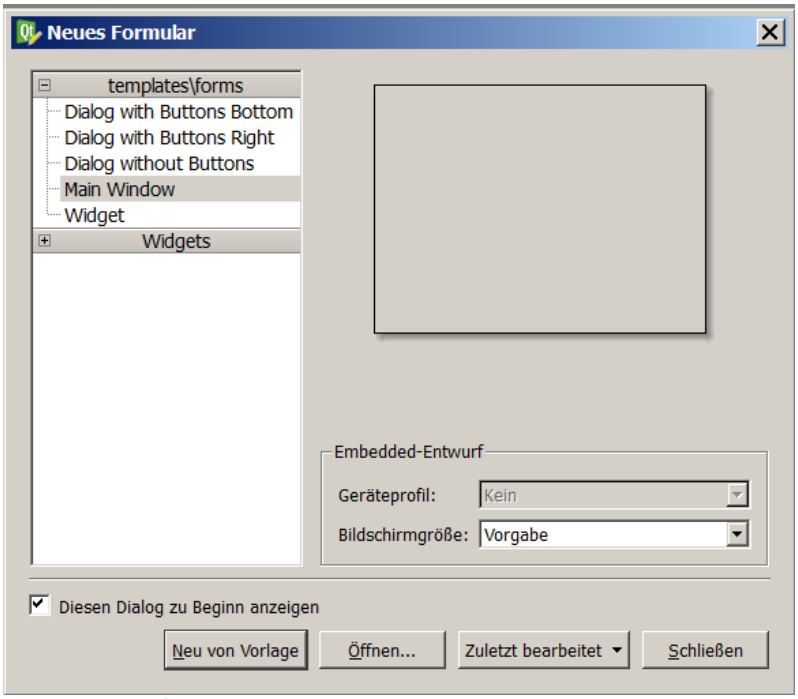

Abbildung 1: Start des designers

Auswahl des Templates:

- *Dialog:* ein GUI, das einen fixen Prozess (accept/rejected) unterstützt
- *Widget:* ein einfacher Container
- *Main Window:* ein Widget mit einer Menubar

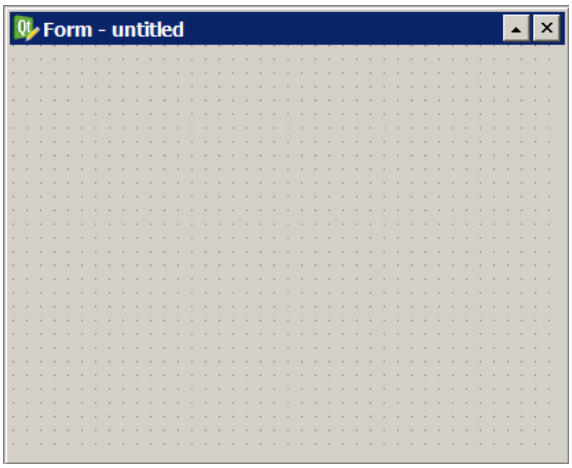

Abbildung 2: Widget

Ein Widget kann weitere Container aufnehmen. Mithilfe von Layouts können die grafischen Komponenten platziert werden.

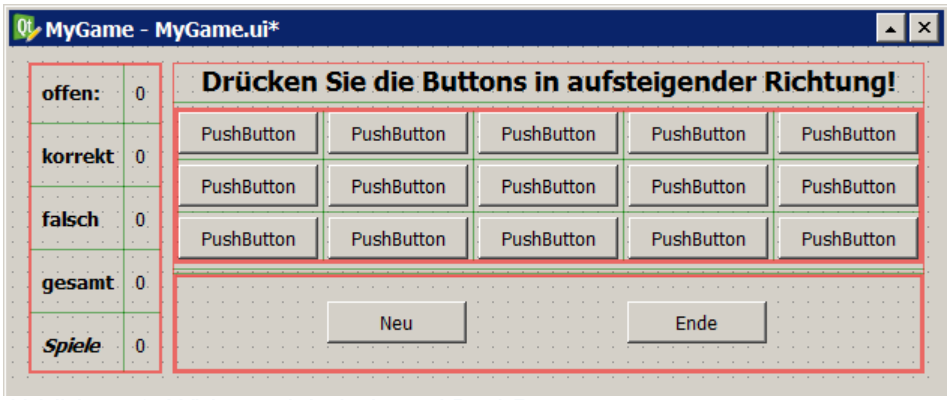

Abbildung 3: Widget mit Labels und PushButtons

Die wichtigsten Layouts sind:

- *Vertical layout:* vertikale Anordnung der Komponenten
- *Horizontal layout:* horizontale Anordnung der Komponenten
- *Grid layout:* Anordnung der Komponenten in Zeilen und Spalten
- *Form layout:* Anordnung der Komponenten in 2 Spalten
  links das Label – rechts das entsprechende Eingabe-Widget

Die Komponenten können im rechten Menü editiert werden.

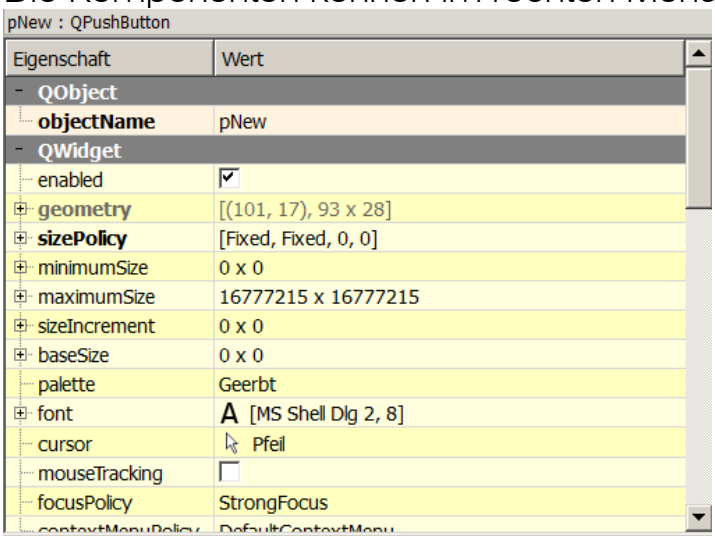

Abbildung 4: PushButton Neu

Neben der Bezeichnung (text) kann der ObjectName definiert werden, welcher später als Referenz verfügbar wird.
Es gibt viele weitere Einstellungsmöglichkeiten für das Widget.
Wichtig ist hier die SizePolicy, welche festlegt, wie bei Größenänderungen des umschließenden Widgets verfahren werden soll.

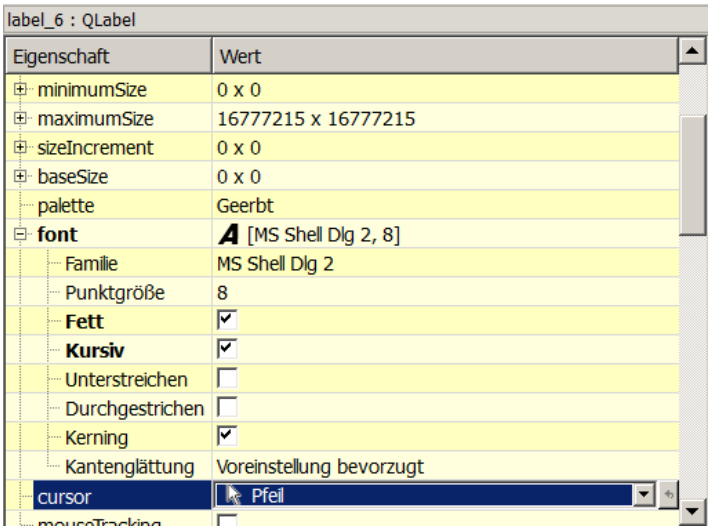

Abbildung 5: QLabel Spiele

1.2.3. Signale und Slots

Das Qt-Framework besitzt ein Eventhandling, dass dem Observer-Pattern nachempfunden ist. Der Sender (Observable) ruft bei Eintritt eines speziellen Signals (Events) bei Empfängern (Observer) eine spezielle Methode oder Callable (Slot) auf.

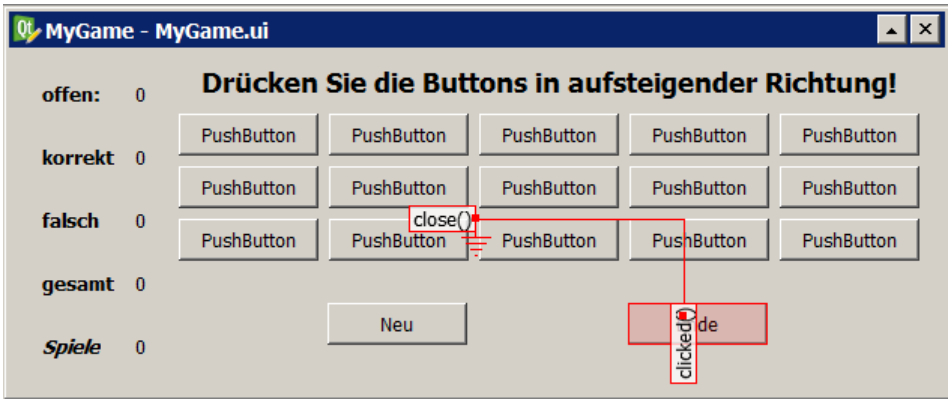

Abbildung 6: Signale und Slots bearbeiten

In diesem Beispiel soll bei Click des Ende-PushButtons das Widget MyGame beendet werden.

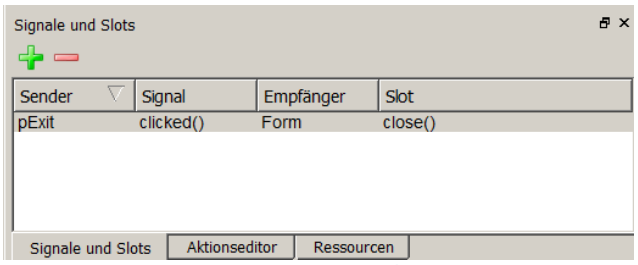

Abbildung 7: PushButton pExit: Signal clicked()

Das Eventhandling kann neben der grafischen Erstellung auch im entsprechenden Menü erstellt und bearbeitet werden.

Leider sind nicht alle Events so einfach umzusetzen, weshalb entsprechende Codierung ebenfalls notwendig sein wird.

1.2.4. Konvertierung in Python

Nach dem Speichern des GUI's als .ui-File muss noch eine Konvertierung in Python angestoßen werden.

1.2.4.1. PyQT

Der Converter befindet sich im Scripts-Verzeichnis

```
pyuic5 -o MyView.py MyGame.ui
```

Mit dem Parameter –o wird das Output-File für Windows definiert.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'MyGame.ui'
#
# Created: Sun Dec  7 17:20:10 2014
#      by: PyQt5 UI code generator 5.2.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(619, 229)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Ignored,
                                           QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(Form.sizePolicy().hasHeightForWidth())
        Form.setSizePolicy(sizePolicy)
```

Achtung: Bei einer neuerlichen Konvertierung gehen alle Änderung im Code verloren!

```
        self.pExit.clicked.connect(Form.close)
```

Neben dem Erstellen und dem Konfigurieren der Komponenten ist in der Methode *setupUi* auch das Eventhandling des Ende-Buttons umgesetzt.

```
    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "MyGame"))
        self.lSum.setText(_translate("Form", "0"))
        self.label_2.setText(_translate("Form", "offen:"))
        self.label_4.setText(_translate("Form", "falsch"))
        self.label_5.setText(_translate("Form", "gesamt"))
        self.pNew.setText(_translate("Form", "Neu"))
        self.pExit.setText(_translate("Form", "Ende"))
```

In der zweiten Methode *retranslateUi* werden die textuellen Einstellungen der geänderten Komponenten umgesetzt.

1.2.4.2. PySide

Der Converter befindet sich ebenfalls im Scripts-Verzeichnis

```
pyside-uic -o MyView.py MyGame.ui
```

Die Umsetzung für PySide ist wieder sehr analog zu PyQT5

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'MyGame.ui'
#
# Created: Sun Dec 13 21:54:19 2015
#      by: pyside-uic 0.2.15 running on PySide 1.2.2
#
# WARNING! All changes made in this file will be lost!

from PySide import QtCore, QtGui

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(619, 229)
        sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Ignored,
                                       QtGui.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(Form.sizePolicy().hasHeightForWidth())
        Form.setSizePolicy(sizePolicy)

        self.pExit.clicked.connect(Form.close)
```

PyQT in Version 5 verwendet standardgemäß UTF-8-Kodierung. In Qt 4.x ist jedoch Latin-1 (ASCII-Kodierung) default. Strings müssen deshalb als Unicode UTF-8 gekennzeichnet werden (`QtGui.QApplication.UnicodeUTF8`)

```
    def retranslateUi(self, Form):
        _translate = QtGui.QApplication.translate
        Form.setWindowTitle(_translate("Form", "MyGame", None,
             QtGui.QApplication.UnicodeUTF8))
        self.lSum.setText(_translate("Form", "0", None,
             QtGui.QApplication.UnicodeUTF8))
        self.label_2.setText(_translate("Form", "offen:" , None,
             QtGui.QApplication.UnicodeUTF8))
        self.label_4.setText(_translate("Form", "falsch", None,
             QtGui.QApplication.UnicodeUTF8))
        self.label_5.setText(_translate("Form", "gesamt", None,
             QtGui.QApplication.UnicodeUTF8))
        self.pNew.setText(_translate("Form", "Neu", None,
             QtGui.QApplication.UnicodeUTF8))
```

```
        self.pExit.setText(_translate("Form", "Ende", None,
             QtGui.QApplication.UnicodeUTF8))
```

1.2.5. MVC-Pattern

1.2.5.1. PyQt

Wie o.a. ändert sich die View-Klasse in Python mit Änderungen im GUI. Ebenso sollte auch das Model nur lose gekoppelt sein.

Vorschlag für eine Controller-Klasse:

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
import MyView, MyModel


class MyController(QWidget):
    """ MVC pattern: Creates a controller - mvc pattern.

    """
    def __init__(self, parent=None):
        """ Create a new controller with a object MyView and a object MyModel
        using the mvc pattern.

        :param parent:
        """
        super().__init__(parent)
        self.myForm = MyView.Ui_Form()
        self.myForm.setupUi(self)

        self.myModel = MyModel.MyModel()

        # connect the buttons with the clicked signal
        self.connectButtons()
        # start a new game
        self.start()
```

1.2.5.2. PySide

Wie o.a. ändert sich die View-Klasse in Python mit Änderungen im GUI. Die Änderungen für PySide sind nur gering

Vorschlag für eine Controller-Klasse:

```
from PySide.QtCore import *
from PySide.QtGui import *
import MyView, MyModel


class MyController(QWidget):
    """ MVC pattern: Creates a controller - mvc pattern.

    """
    def __init__(self, parent=None):
        """ Create a new controller with a object MyView and a object MyModel
```

```
        using the mvc pattern.

        :param parent:
        """
        super().__init__(parent)
        self.myForm = MyView.Ui_Form()
        self.myForm.setupUi(self)

        self.myModel = MyModel.MyModel()

        # connect the buttons with the clicked signal
        self.connectButtons()
        # start a new game
        self.start()
```

Die Applikation kann entweder im Controller oder in einem weiteren Python-File erfolgen:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    c = MyController()
    c.show()
    sys.exit(app.exec_())
```