PyQT_Primitive_1_App_V03

December 12, 2023

0.1 Purpose: Use primitive PyQt-App with QtAgg-backend for Matplotlib

Basic idea: Box with a row for buttons, a central panel for some action, and a raow with two areas for print output

A red "laser beam" will shoot a the corners of the panel or at the mouse pointer

0.1.1 Imports

[1]:
```
import time
# import gc  # garbage collection
import sys # for PyQt5
import numpy as np
import queue


import matplotlib.backends
import matplotlib.pyplot as plt
from matplotlib.figure import Figure

from PyQt5 import QtWidgets, QtCore
from PyQt5.QtCore import QSize, Qt, QLine, QPoint
from PyQt5.QtGui import QPainter, QPen, QBrush, QFont, QColor
from PyQt5.QtGui import QTextCursor
from PyQt5.QtWidgets import QMainWindow, QWidget, QGridLayout
from PyQt5.QtWidgets import QPushButton, QLabel, QGroupBox
from PyQt5.QtWidgets import QVBoxLayout, QHBoxLayout
from PyQt5.QtWidgets import QFrame, QSizePolicy, QTextEdit

# redirects of stdout to QTextEdit
# from contextlib import redirect_stdout
```

0.1.2 Activate QtAgg-backend !!! Do NOT forget !!!

[2]:
```
matplotlib.use('QtAgg')
```

[ ]:
```

```

0.2 Code for nested widgets

0.2.1 Code for innermost widget: Defines a Panel for action

This panel will be framed and raised above background

This is where the "shooting" takes place

[3]:
```
class PanelArea(QWidget):

    def __init__(self, parentW
                 , beam_time_interval
                 , num_shoot_iterations
                 , plus_iterations):

        super(PanelArea, self).__init__()

        self.parentW = parentW
        self.beam_time_interval = beam_time_interval

        self.shoot_corners = False
        self.j_target = 0
        self.shooting_active = False
        self.num_shoot_iterations = num_shoot_iterations
        self.initial_num_shoot_iterations = num_shoot_iterations
        self.plus_iterations = plus_iterations

        # Track Mouse Movement
        self.setMouseTracking(True)
        self.mouse_x = 0
        self.mouse_y = 0

        # control changes of window size
        self.width_old = self.width()
        self.width_old = self.height()
        self.x_leaving = 0
        self.y_leaving = 0
        self.myheight = self.height()
        self.mywidth = self.width()

        # Parameters for central ellipse
        self.b_draw_ell = True
        self.ell_axis1 = 75
        self.ell_axis2 = 35

        # Laser beam
        self.b_plot_line = False
        self.b_rupt_line = False
```

```
        self.line = QLine()

        # Pens, brushes for drawing elements
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        # line drawing
        self.pen_line = QPen(Qt.red, 3)
        # ellipse border
        # filling the ellipse
        self.pen_ell = QPen(Qt.black, 5)
        darkred = QColor(120, 0, 0)
        self.brush_ell = QBrush(darkred)


        # Draw the ellipse 1st time
        # ~~~~~~~~~~~~~~~~~~~~~~~~
        self.update()


    # Function to trigger beam to panel corners
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def draw_line_to_corners(self):
        self.shoot_corners = True
        self.b_rupt_line = False

        if self.shooting_active:
            self.num_shoot_iterations += self.plus_iterations
            self.parentW.parentW.print_addedShots()
        else:
            self.draw_line()


    # Function to trigger beam to mouse position
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def draw_line_to_mouse(self):
        self.shoot_corners = False
        self.b_rupt_line = True
        if self.shooting_active:
            self.num_shoot_iterations += self.plus_iterations
            self.parentW.parentW.print_addedShots()
        else:
            self.draw_line()


    # Control frequency of laser beam + number of "shots"
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def draw_line(self):
        self.shooting_active = True
```

```
        # Loop for "shooting"
        # ~~~~~~~~~~~~~~~~~~~
        # Need while loop because we will
        # change "num_shoot_iterations" dynamically
        i = 0
        while i < self.num_shoot_iterations:
            # We have to drop painting peridically
            # for blinking effect
            if i%2 == 0:
                self.b_plot_line = True
            else:
                self.b_plot_line = False

            # Trigger a paintEvent and direct painting
            # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            self.update()
            QtWidgets.QApplication.processEvents()

            # raise iterator variable + printing
            # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            # --- if shooting is not interrupted
            if not self.b_rupt_line:
                #print("i = ", i)
                i += 1
                # print shot reserve sometimes
                if i%100 == 0:
                    sr = self.num_shoot_iterations - (i+1)
                    self.parentW.parentW.print_shotReserve(sr, b_clear=False)


            # set frequency
            time.sleep(self.beam_time_interval)

        # Reset parameters
        # ~~~~~~~~~~~~~~~~~~~
        self.b_plot_line = False
        self.shooting_active = False
        self.parentW.parentW.print_shoot_stop()
        self.num_shoot_iterations = self.initial_num_shoot_iterations
        sr = self.num_shoot_iterations
        self.parentW.parentW.print_shotReserve(sr, b_clear=True)
        # comment the next two statements
        # if you want to see the last beam intentionally
        self.update()
        QtWidgets.QApplication.processEvents()
```

```
    # Function to react to mouse movements
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def mouseMoveEvent(self, event):
        ##if event.type() == QtCore.QEvent.MouseMove:
        self.mouse_x = event.x()
        self.mouse_y = event.y()
        self.parentW.parentW.mouse_label.setText(
            'Mouse coords: ( %d : %d )' % (self.mouse_x, self.mouse_y))
        self.b_rupt_line = False

    # Function to interrupt shooting
    # when action panel area is left
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def leaveEvent(self, event):
        ##print("leave event")
        if not self.shoot_corners:
            self.b_rupt_line = True


    # Function to react to update-requests
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def paintEvent(self, event):

        # Short lived painter - see docu
        painter = QPainter(self)

        # !!!!!!
        # Width/height are only calculated
        # within paintEvents during window resizing
        width = self.width()
        height = self.height()

        # Handle intermediate resizing
        # by interrupting beam display
        b_plot_line2 = True

        xc = int(width/2)
        yc = int(height/2)

        # standard target coordinates
        # - can be overwritten below
        xt = self.mouse_x
        yt = self.mouse_y

        if width != self.width_old or \
```

```
           height != self.height_old:
            b_plot_line2 = False
            xt = xc
            yt = yc

        #print("x,y:", y, x)
        self.center = QPoint(xc, yc)

        # Paint the laser beam (= line)
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        # Setting painter properties
        painter.setPen(self.pen_line)

        if self.b_plot_line and b_plot_line2 \
        and not self.b_rupt_line :

            if self.shoot_corners:
                # set new target coordinates
                if self.j_target == 0:
                    xt = 0; yt = 0
                elif self.j_target == 1:
                    xt = self.width(); yt = 0
                elif self.j_target == 2:
                    xt = self.width(); yt = self.height()
                elif self.j_target == 3:
                    xt = 0; yt = self.height()

                if self.j_target == 3:
                    self.j_target = 0
                else:
                    self.j_target += 1

            self.target = QPoint(xt, yt)
            self.line = QLine(self.target, self.center)
            painter.drawLine(self.line)

        # Paint the central ellipse
        # ~~~~~~~~~~~~~~~~~~~~~~~~
        # (brush needed to fill out area)
        painter.setPen(self.pen_ell)
        painter.setBrush(self.brush_ell)

        if self.b_draw_ell:
            painter.drawEllipse(QPoint(self.center)
                                , self.ell_axis1
                                , self.ell_axis2)
```

```
        self.width_old = width
        self.height_old = height
        self.mywidth = width
        self.myheight = height

    # Function to stop loop before closing
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def closeEvent(self, event):
        self.num_shoot_iterations = 0
        time.sleep(0.05)
        event.accept()
```

0.2.2 Code for a widget that draws a frame around the innermost panel area

This widget becomes a parent of the innermost widget

[4]:
```
class PanelFrame(QFrame):

    def __init__(self, parentW, beam_timeinterval=0.05
                 , num_shoot_iterations=401
                 , plus_iterations=200):

        super(PanelFrame, self).__init__()

        self.parentW = parentW

        # Content: An instance of class Panel
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        # Area gets "self" as parentW
        self.inner_area_widget = PanelArea(self
                                       , beam_timeinterval
                                       , num_shoot_iterations
                                       , plus_iterations)
        self.frame_layout = QVBoxLayout()
        self.setLayout(self.frame_layout)
        self.frame_layout.addWidget(self.inner_area_widget)

        self.setFrameShape(QFrame.Panel)
        self.setFrameShadow(QFrame.Raised)
        self.setLineWidth(20)
        #self.setStyleSheet("background-color: #302010; ")

        # No mouse-tracking required
        self.setMouseTracking(False)
```

0.2.3 Code for the outermost Qt-window and a container for the other widgets

A central widget of the Qt-window becomes the parent of all other widgets

We define three rows in a vertical layout

The widget for the central row-layout becomes a container for the frame/panel

[5]:
```
class MainWindow(QtWidgets.QMainWindow):

    def __init__(self, shoot_interval
                 , num_shoot_iterations = 401
                 , plus_iterations = 200):
        QtWidgets.QMainWindow.__init__(self)

        self.setWindowTitle("Laser")

        # Parameters (transfer to child objects)
        # ~~~~~~~~~~~
        # timer for "laser beam" blinking
        self.beam_time_interval = shoot_interval
        # num of laser shots
        self.num_shoot_iterations = num_shoot_iterations
        self.plus_iterations = plus_iterations

        # Main inner window = Container (simple QWidget)
        # ~~~~~~~~~~~~~~~~~~
        self.inner_win = QtWidgets.QWidget()
        self.setCentralWidget(self.inner_win)

        self.inwin_layout = QtWidgets.QVBoxLayout()
        self.inner_win.setLayout(self.inwin_layout)
        self.inwin_layout.setContentsMargins(0,0,0,0)
        self.inwin_layout.setSpacing(20)

        # Set initial size of the Qt-window
        # ~~~~~~~~~~~~~~~~
        self.resize(750, 900)

        # Preparations for contents
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~
        # just for admin purposes
        self.d_mywidgets = {}
        # colors
        self.col_red = QColor('red')
        self.col_darkred = QColor(125, 0, 0)
        self.col_darkblue = QColor(0, 0, 125)
```

```
        self.col_darkgreen = QColor(0, 125, 0)
        self.col_black = QColor(3,3,3)

        # Create contents in the central widget
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        self.create_window_contents()

        # no mouse-tracking required on this level
        self.setMouseTracking(False)

        # show() (non-blocking for notebook cells)
        # ~~~~~~
        # (this is not Matplotlibs plt.show)
        self.show()

    # Fill the central widget with contents
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def create_window_contents(self):

        # A groupbox for multiple buttons
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        self.groupbox1 = QGroupBox("  Main messages")
        self.groupbox1.setStyleSheet('font-weight:bold;')
        self.groupbox1.setFixedHeight(100)
        self.inwin_layout.addWidget(self.groupbox1)
        self.vbox1 = QHBoxLayout()
        self.groupbox1.setLayout(self.vbox1)

        # 1-st button: shoot beam periodically at panel's corners
        self.but_shoot_corners = QPushButton('shoot\ncorners', self)
        #self.but_shoot_corners.setMinimumWidth = 100
        self.but_shoot_corners.setMinimumSize(QSize(100, 50))
        sizePolicy_but = QSizePolicy(QSizePolicy.Maximum, QSizePolicy.Maximum)
        self.but_shoot_corners.setSizePolicy(sizePolicy_but)
        self.vbox1.addWidget(self.but_shoot_corners)

        # 2-nd button: Shoot beam at mouse pointer position
        self.but_shoot_mouse = QPushButton('shoot\nmouse', self)
        #self.but_shoot_corners.setMinimumWidth = 100
        self.but_shoot_mouse.setMinimumSize(QSize(100, 50))
        sizePolicy_but = QSizePolicy(QSizePolicy.Maximum, QSizePolicy.Maximum)
        self.but_shoot_mouse.setSizePolicy(sizePolicy_but)
        self.vbox1.addWidget(self.but_shoot_mouse)


        # 3-rd button: Erase button 2 and ellipse
        self.but_kill_mouse = QPushButton('kill mouse\nshooter', self)
```

```
        #self.but_shoot_corners.setMinimumWidth = 100
        self.but_kill_mouse.setMinimumSize(QSize(120, 50))
        sizePolicy_but = QSizePolicy(QSizePolicy.Maximum, QSizePolicy.Maximum)
        self.but_kill_mouse.setSizePolicy(sizePolicy_but)
        self.vbox1.addWidget(self.but_kill_mouse)

        self.b_1st_click_but_kill = False

        # Stretch element
        self.vbox1.insertStretch(3)

        # Text label to show mouse position inside panel
        self.mouse_label = QLabel(self) # provide self to the QFrame
        self.mouse_label.setFrameStyle(QFrame.Panel | QFrame.Sunken)
        #self.label.setText('Mouse coords: ( %d : %d )' % (event.x(), event.
↪y()))
        self.mouse_label.setText('Mouse coords: ( %d : %d )' % (0.0, 0.0))
        self.mouse_label.setMinimumSize(QSize(230, 50))
        sizePolicy_label = QSizePolicy(QSizePolicy.Maximum, QSizePolicy.Maximum)
        self.mouse_label.setSizePolicy(sizePolicy_label)
        self.vbox1.addWidget(self.mouse_label)

        # self.inwin_layout.setStretchFactor(self.groupbox1, 1)


        # A (raised) Panel and Frame for action
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        self.groupbox2 = QGroupBox("  Panel")
        self.groupbox2.setStyleSheet('font-weight:bold;')
        #self.groupbox2.resize(500,400)
        self.inwin_layout.addWidget(self.groupbox2)
        self.vbox2 = QVBoxLayout()
        self.groupbox2.setLayout(self.vbox2)

        # Initialize an Areaframe - self is provided as parameter parentW
        # The frame will load and contain a panel
        self.frame1 = PanelFrame(self
                                 , self.beam_time_interval
                                 , num_shoot_iterations
                                 , plus_iterations)
        self.vbox2.addWidget(self.frame1)
        #self.inwin_layout.setStretchFactor(self.groupbox2, 4)


        # A groupbox Panel for info prints
        # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        # Add a Qtext
```

```
        self.groupbox3 = QGroupBox("  Infos")
        self.groupbox3.setStyleSheet('font-weight:bold;')
        #self.groupbox2.resize(500,400)
        self.groupbox3.setFixedHeight(260)
        self.inwin_layout.addWidget(self.groupbox3)
        self.vbox3 = QHBoxLayout()
        self.groupbox3.setLayout(self.vbox3)
        self.textedit1 = QTextEdit()
        self.vbox3.addWidget(self.textedit1)
        self.textedit2 = QTextEdit()
        self.vbox3.addWidget(self.textedit2)
        self.textedit2.setFontWeight(QFont.Normal)
        self.textedit2.setTextColor(self.col_black)

        # Set button actions
        # ~~~~~~~~~~~~~~~~~~
        self.bind_buts_to_callbacks()

    # Bind user actions
    # ~~~~~~~~~~~~~~~~~
    def bind_buts_to_callbacks(self):
        self.but_shoot_corners.clicked.connect(self.
↪print_clearStatusChange_Corners)
        self.but_shoot_corners.clicked.connect(self.frame1.inner_area_widget.
↪draw_line_to_corners)
        self.but_shoot_mouse.clicked.connect(self.print_clearStatusChange_Mouse)
        self.but_shoot_mouse.clicked.connect(self.frame1.inner_area_widget.
↪draw_line_to_mouse)
        self.but_kill_mouse.clicked.connect(self.kill_mouse_shooter)


    # Function to remove mouse_shooting
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def kill_mouse_shooter(self):
        if not self.b_1st_click_but_kill:
            self.but_shoot_mouse.hide()
            if not self.frame1.inner_area_widget.shoot_corners:
                self.frame1.inner_area_widget.num_shoot_iterations = False
                self.frame1.inner_area_widget.b_plot_line = False
            self.frame1.inner_area_widget.b_draw_ell = False
            self.b_1st_click_but_kill = True
        else:
            self.but_shoot_mouse.setVisible(True)
            self.frame1.inner_area_widget.b_draw_ell = True
            self.frame1.inner_area_widget.update()
            QtWidgets.QApplication.processEvents()
            self.b_1st_click_but_kill = False
```

```
    # Function to write Output when changing to corner-shooting
    # ~~~~~~~~~~~~~~~~~~~~~~~~~
    def print_clearStatusChange_Corners(self):
        #print("her")
        font_act = self.textedit1.currentFont()
        text = "Status changed: Shoot at Corners\n"
        self.textedit1.moveCursor(QTextCursor.End)
        #f_act = self.textedit1.currentFont()
        # must be set after Cursor is moved
        self.textedit1.setFontWeight(QFont.Bold)
        self.textedit1.setTextColor(self.col_darkred)
        self.textedit1.insertPlainText(text)
        QtWidgets.QApplication.processEvents()
        #self.textedit1.setCurrentFont(font_act)


    # Function to write Output when changing to mouse-shooting
    # ~~~~~~~~~~~~~~~~~~~~~~~~~
    def print_clearStatusChange_Mouse(self):
        #print("her")
        font_act = self.textedit1.currentFont()
        text = "Status changed: Shoot at Mouse\n"
        self.textedit1.moveCursor(QTextCursor.End)
        #f_act = self.textedit1.currentFont()
        # must be set after Cursor is moved
        self.textedit1.setFontWeight(QFont.Bold)
        self.textedit1.setTextColor(self.col_darkblue)
        self.textedit1.insertPlainText(text)
        QtWidgets.QApplication.processEvents()


    # Function to write Infos about added shots
    # ~~~~~~~~~~~~~~~~~~~~~~~~~
    def print_addedShots(self):
        text = "Added shots: " + str(self.plus_iterations) + "\n"
        self.textedit1.moveCursor(QTextCursor.End)
        self.textedit1.setFontWeight(QFont.Normal)
        self.textedit1.setTextColor(self.col_black)
        self.textedit1.insertPlainText(text)
        #self.textedit1.append( text )
        QtWidgets.QApplication.processEvents()


    # Function to write Infos about shot reserve replenishing
    # ~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
    def print_shotReserve(self, shotReserve, b_clear=False):
        if b_clear:
            self.textedit2.clear()
            text_stop = "Shooting stopped. Reserve replenished" + "\n"
            self.textedit2.moveCursor(QTextCursor.End)
            self.textedit2.setFontWeight(QFont.Normal)
            self.textedit2.setTextColor(self.col_black)
            self.textedit2.insertPlainText(text_stop)

        text = "Shot reserve: " + str(shotReserve) +"\n"
        self.textedit2.moveCursor(QTextCursor.End)
        self.textedit2.setFontWeight(QFont.Normal)
        self.textedit2.setTextColor(self.col_black)
        self.textedit2.insertPlainText(text)
        #self.textedit1.append( text )
        QtWidgets.QApplication.processEvents()

    # Function to write Infos about stopped shooting
    # ~~~~~~~~~~~~~~~~~~~~~~~~~
    def print_shoot_stop(self):
        text = "Shooting stopped" + "\n"
        self.textedit1.moveCursor(QTextCursor.End)
        self.textedit1.setFontWeight(QFont.Normal)
        self.textedit1.setTextColor(self.col_darkgreen)
        self.textedit1.insertPlainText(text)
        #self.textedit1.append( text )
        QtWidgets.QApplication.processEvents()


    # Function to stop loop before killing the app window
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    def closeEvent(self, event):
        self.frame1.inner_area_widget.num_shoot_iterations = 0
        print("User is closing Main Win ")
        # just to be on the safe side
        time.sleep(0.1)
        # required !
        event.accept()
```

0.2.4 Start the action

[6]:
```
shoot_interval = 0.05
num_shoot_iterations = 401
plus_iterations = 200
mainWin = MainWindow(shoot_interval, num_shoot_iterations, plus_iterations)
```

```
User is closing Main Win
```

[ ]: