```python
#plugin_viewer.py

# coding: utf-8

from PyQt5 import QtCore, QtGui, QtWidgets, QtOpenGL
import math


class Viewer(QtWidgets.QGraphicsView):
    # color of frame background
    backgroundColor = QtGui.QColor(31, 31, 47)
    # color of frame border
    borderColor = QtGui.QColor(58, 58, 90)

    sig_position = QtCore.pyqtSignal(tuple)

    def __init__(self, parent=None, *args, **kwargs):
        super().__init__(parent=parent, *args, **kwargs)

        self.setBackgroundBrush(self.backgroundColor)  # set background color

        self.setOptimizationFlag(self.DontSavePainterState)

        self.setRenderHints(
            QtGui.QPainter.Antialiasing | QtGui.QPainter.TextAntialiasing | QtGui.QPainter.SmoothPixmapTransform
        )

        if QtOpenGL.QGLFormat.hasOpenGL():
            self.setRenderHint(QtGui.QPainter.HighQualityAntialiasing)

        self.setResizeAnchor(self.AnchorUnderMouse)  # set the current position of mouse as anchor

        self.setRubberBandSelectionMode(QtCore.Qt.IntersectsItemShape)

        self.setTransformationAnchor(self.AnchorUnderMouse)

        self.setViewportUpdateMode(self.SmartViewportUpdate)

        self._scene = QtWidgets.QGraphicsScene(self)
        self.setScene(self._scene)

        self._pix_item = QtWidgets.QGraphicsPixmapItem()
        self._scene.addItem(self._pix_item)

        # self.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        # self.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)


self.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContentsOnFirstShow)
        self.setAlignment(QtCore.Qt.AlignJustify | QtCore.Qt.AlignVCenter | QtCore.Qt.AlignHCenter)

        # enable zoom in
        self.setAcceptDrops(True)
        self.setDragMode(QtWidgets.QGraphicsView.NoDrag)

        self.last = "Click"

        self.canDraw = False
```

```
        self.item_rect = QtWidgets.QGraphicsRectItem()
        self.scene().addItem(self.item_rect)
        self.item_rect.setBrush(QtGui.QBrush(QtGui.QColor(255, 99, 45, 60)))
        self.item_rect.setPen(QtGui.QColor(109, 25, 75))

        self.item_text = QtWidgets.QGraphicsTextItem()
        # self.item_text.setPlainText('hello world')
        self.scene().addItem(self.item_text)

        self.setAllowDrawRect(False)

        self.begin = QtCore.QPoint()
        self.end = QtCore.QPoint()
        self.point_a = 0, 0
        self.point_b = 0, 0

    def wheelEvent(self, event: QtGui.QWheelEvent) -> None:
        """press 'ctrl' with wheel event to zoom in/out"""
        if event.modifiers() & QtCore.Qt.ControlModifier:
            self.scaleView(math.pow(2.0, event.angleDelta().y() / 240.0))
            return event.accept()
        super(QtWidgets.QGraphicsView, self).wheelEvent(event)

    def scaleView(self, scaleFactor):
        factor = self.transform().scale(scaleFactor,
scaleFactor).mapRect(QtCore.QRectF(0, 0, 1, 1)).width()
        if factor < 0.07 or factor > 100:
            return
        self.scale(scaleFactor, scaleFactor)

    def display_pix(self, pix: QtGui.QPixmap):
        """update the pic to display"""
        if not isinstance(pix, QtGui.QPixmap):
            raise TypeError
        self._pix_item.setPixmap(pix)
        w, h = pix.width(), pix.height()
        self.fitInView(QtCore.QRectF(0, 0, w, h), QtCore.Qt.KeepAspectRatio)
        self._scene.update()

    def clear_pix(self):
        """clear out the pic"""
        self._pix_item.setPixmap(QtGui.QPixmap())

    def mousePressEvent(self, event: QtGui.QMouseEvent) -> None:
        self.last = "Click"
        if self.canDraw and event.buttons() & QtCore.Qt.LeftButton:
            self.begin = event.pos()
            self.end = event.pos()
            p = self._pix_item.mapFromScene(self.mapToScene(event.pos()))
            self.point_a = self.point_b = p.x(), p.y()

    def mouseMoveEvent(self, event: QtGui.QMouseEvent) -> None:
        self.end = event.pos()
        if self.canDraw and (event.buttons() & QtCore.Qt.LeftButton):
            rect = QtCore.QRectF(self.mapToScene(self.begin),
self.mapToScene(self.end))
            self.item_rect.setRect(rect)
            self.item_text.setPos(self.mapToScene(self.begin))
            p = self._pix_item.mapFromScene(self.mapToScene(event.pos()))
            self.point_b = p.x(), p.y()

            x1, y1 = self.point_a
```

```
            x2, y2 = self.point_b

            w = x2 - x1
            h = y2 - y1

            if w == 0 or h == 0:
                text_pos = 0, 0, 0, 0
            elif w > 0 and h > 0:
                text_pos = x1, y1, w, h
            elif w > 0 and h < 0:
                text_pos = x1, y2, w, -h
            elif w < 0 and h > 0:
                text_pos = x2, y1, -w, h
            elif w < 0 and h < 0:
                text_pos = x2, y2, -w, -h
            else:
                text_pos = 0, 0, 0, 0


self.item_text.setPos(self._pix_item.mapToScene(QtCore.QPointF(text_pos[0], text_pos[1])))

            self.sig_position.emit(text_pos)

    def mouseReleaseEvent(self, event: QtGui.QMouseEvent) -> None:
        if self.last == "Click":
            print("click")
        else:
            print("Double Click")

    def mouseDoubleClickEvent(self, event: QtGui.QMouseEvent) -> None:
        self.last = "Double Click"

    def setAllowDrawRect(self, b: bool):
        """"""
        if b is True:
            # self.item_rect.show()
            self.canDraw = True
        else:
            # self.item_rect.hide()
            self.canDraw = False

    def display_text(self, txt: str, font: QtGui.QFont):
        self.item_text.setFont(font)
        self.item_text.setPlainText(txt)

    def clear_text(self):
        self.item_text.setPlainText("")

    def hide_rect(self):
        self.item_rect.hide()

    def display_rect(self):
        self.item_rect.show()

    def save_image(self) -> QtGui.QPixmap:
        size = self._pix_item.boundingRect().size()
        pix = QtGui.QPixmap(QtCore.QSize(int(size.width()), int(size.height())))
        painter = QtGui.QPainter(pix)
        painter.setRenderHint(QtGui.QPainter.Antialiasing, True)
        self.scene().render(
```

```
177             painter, self._pix_item.boundingRect(), self._pix_item.boundingRect(),
    QtCore.Qt.KeepAspectRatio,
178         )
179         painter.end()
180         return pix
181
```