Actividad 09 - QScene

Arturo Sánchez Sánchez

Seminario de Algoritmia

Lineamientos de evaluación

- El reporte está en formato Google Docs o PDF.
- El reporte sigue las pautas del Formato de Actividades .
- El reporte tiene desarrollada todas las pautas del Formato de Actividades.
- Se muestra captura de pantalla de lo que se pide en el punto 2.

Desarrollo

Toma capturas de pantalla de la ejecución mostrando la visualización de al menos 5 partículas en el QScene

Conclusiones

Una actividad muy interesante en su totalidad, no conocía este tipo de herramientas pero creo que tiene un capacidad de uso muy grande como lo es el dibujo y el trazado de diferentes cosas, nos adentramos a un mundo de posibilidades una vez que

sabemos usar las herramientas que nos da python para el uso de la programación orientada a objetos, espero que en un futuro podamos seguir trabajando con esto, creando cada vez más cosas teniendo como límite únicamente nuestra imaginación.

Referencias

MICHEL DAVALOS BOITES. (2020, 29 octubre). *PySide2 - QTableWidget (Qt for Python)(V)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=1yEpAHaiMxs

Código

mainwindow.py

```
from turtle import color
from PySide2.QtWidgets import QMainWindow, QFileDialog, QMessageBox,
QTableWidgetItem, QGraphicsScene
from PySide2.QtCore import Slot
from PySide2.QtGui import QPen, QColor, QTransform
from ui_mainwindow import Ui_MainWindow
from particulasact.particula import Particula
from particulasact.index import Nodo, Lista_ligada



class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.lista_ligada = Lista_ligada()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.agregarFinal_pushButton.clicked.connect(
            self.click_agregarFinal)
        self.ui.agregarInicio_pushButton.clicked.connect(
            self.click_agregarInicio)
        self.ui.mostrar_pushButton.clicked.connect(self.click_mostrar)
        self.ui.actionAbrir.triggered.connect(self.action_abrir_archivo)
        self.ui.actionGuardar.triggered.connect(self.action_guardar_archivo)
        self.ui.mostrar_tabla_pushButton.clicked.connect(self.mostrar_tabla)
        self.ui.buscar_pushButton.clicked.connect(self.buscar_particula)
        self.ui.dibujar.clicked.connect(self.dibujar)
```

```
        self.ui.limpiar.clicked.connect(self.limpiar)
        self.scene = QGraphicsScene()
        self.ui.graphicsView.setScene(self.scene)

    def wheelEvent(self, event):
        if event.delta() > 0:
            self.ui.graphicsView.scale(1.2, 1.2)
        else:
            self.ui.graphicsView.scale(.8, .8)

    @Slot()
    def dibujar(self):
        pen = QPen()
        pen.setWidth(2)
        for particula in self.lista_ligada:
            color = QColor(particula.red, particula.green, particula.blue)
            pen.setColor(color)
            self.scene.addEllipse(particula.origen_x,
                                  particula.origen_y, 5, 5, pen)
            self.scene.addEllipse(particula.destino_x,
                                  particula.destino_y, 5, 5, pen)
            self.scene.addLine(particula.origen_x+3, particula.origen_y+3,
                               particula.destino_x+3, particula.destino_y+3,
pen)

    @Slot()
    def limpiar(self):
        self.scene.clear()
```

```
def creadorDeParticulas(self):
    id = self.ui.id_lineEdit.text()
    destinoX = self.ui.destinoX_spinBox.value()
    origenX = self.ui.origenX_spinBox.value()
    destinoY = self.ui.destinoY_spinBox.value()
    origenY = self.ui.origenY_spinBox.value()
    velocidad = self.ui.velocidad_spinBox.value()
    red = self.ui.red_spinBox.value()
    green = self.ui.green_spinBox.value()
    blue = self.ui.blue_spinBox.value()
    return Particula(id, origenX, origenY,
                     destinoX, destinoY, velocidad, red, green, blue)

def creadorDeRows(self, particula, row):
    id_widget = QTableWidgetItem(str(particula.id))
    origen_x_widget = QTableWidgetItem(str(particula.origen_x))
    destino_x_widget = QTableWidgetItem(str(particula.destino_x))
    origen_y_widget = QTableWidgetItem(str(particula.origen_y))
    destino_y_widget = QTableWidgetItem(str(particula.destino_y))
    velocidad_widget = QTableWidgetItem(str(particula.velocidad))
    red_widget = QTableWidgetItem(str(particula.red))
    green_widget = QTableWidgetItem(str(particula.green))
    blue_widget = QTableWidgetItem(str(particula.blue))
    distancia_widget = QTableWidgetItem(str(particula.distancia))

    self.ui.table.setItem(row, 0, id_widget)
    self.ui.table.setItem(row, 1, origen_x_widget)
```

```
        self.ui.table.setItem(row, 2, destino_x_widget)
        self.ui.table.setItem(row, 3, origen_y_widget)
        self.ui.table.setItem(row, 4, destino_y_widget)
        self.ui.table.setItem(row, 5, velocidad_widget)
        self.ui.table.setItem(row, 6, red_widget)
        self.ui.table.setItem(row, 7, green_widget)
        self.ui.table.setItem(row, 8, blue_widget)
        self.ui.table.setItem(row, 9, distancia_widget)

    @Slot()
    def action_abrir_archivo(self):
        ubicacion = QFileDialog.getOpenFileName(
            self,
            'Abrir Archivo',
            '.',
            'JSON (*.json)'
        )[0]
        if self.lista_ligada.abrir(ubicacion):
            QMessageBox.information(
                self,
                "Exito",
                "Se pudo crear el archivo" + ubicacion
            )
        else:
            QMessageBox.critical(self, "Error", "El archivo no pudo
crearse")

    @Slot()
```

```
    def action_guardar_archivo(self):
        ubicacion = QFileDialog.getSaveFileName(
            self,
            'Guardar Archivo',
            '.',
            'JSON (*.json)'
        )[0]
        if self.lista_ligada.guardar(ubicacion):
            QMessageBox.information(
                self, "Exito", "Se pudo crear el archivo"+ubicacion)
        else:
            QMessageBox.critical(
                self,
                "Error",
                "El archivo no se pudo crear"
            )


    @Slot()
    def click_mostrar(self):
        self.ui.salida.clear()
        self.ui.salida.insertPlainText(str(self.lista_ligada))


    @Slot()
    def click_agregarFinal(self):
        particula = self.creadorDeParticulas()
        nodo = Nodo(particula)
        self.lista_ligada.agregar_final(nodo)
        self.ui.salida.clear()
```

```
        self.ui.salida.insertPlainText("Agregado al Final")


    @Slot()
    def click_agregarInicio(self):
        particula = self.creadorDeParticulas()
        nodo = Nodo(particula)
        self.lista_ligada.agregar_inicio(nodo)
        self.ui.salida.clear()
        self.ui.salida.insertPlainText("Agregado al Inicio")


    @Slot()
    def mostrar_tabla(self):
        self.ui.table.setColumnCount(10)
        headers = ["ID", "Origen X", "Destino X", "Origen Y",
                   "Destino Y", "Velocidad", "Red", "Green", "Blue",
"Distancia"]
        self.ui.table.setHorizontalHeaderLabels(headers)
        self.ui.table.setRowCount(len(self.lista_ligada))
        row = 0
        for particula in self.lista_ligada:
            self.creadorDeRows(particula, row)
            row += 1


    @Slot()
    def buscar_particula(self):
        id = self.ui.search_lineEdit.text()
        self.ui.table.clear()
        found = False
```

```
        for particula in self.lista_ligada:
            if id == particula.id:
                self.ui.table.setRowCount(1)
                self.ui.table.setColumnCount(10)
                headers = ["ID", "Origen X", "Destino X", "Origen Y",
                           "Destino Y", "Velocidad", "Red", "Green", "Blue",
"Distancia"]
                self.ui.table.setHorizontalHeaderLabels(headers)
                self.creadorDeRows(particula, 0)
                found = True


        if not found:
            QMessageBox.warning(
                self,
                "Atencion",
                f"La particula con el id: {id}, no fue encontrada"
            )
```