Actividad 09 // QScene

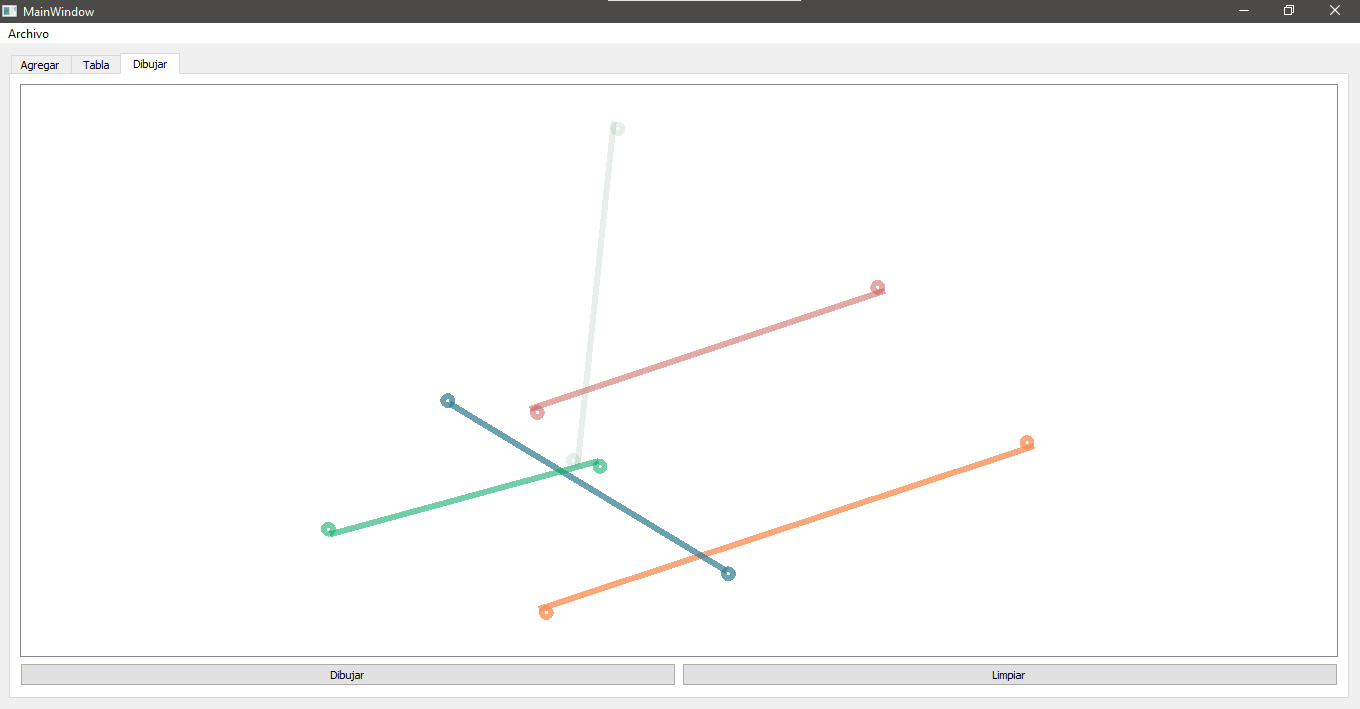
Sámano Juárez Juan Jesus.

Seminario de Solución de Problemas de Algoritmia

**Lineamiento de evaluación.**

* [ ] El reporte está en formato Google Docs o PDF.
* [ ] El reporte sigue las pautas del [Formato de Actividades](https://www.notion.so/Formato-de-Actividades-d098e1ce3a62423ca47c3661c5d93444) .
* [ ] El reporte tiene desarrollada todas las pautas del [Formato de Actividades](https://www.notion.so/Formato-de-Actividades-d098e1ce3a62423ca47c3661c5d93444).
* [ ] 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.

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media**5 partículas en tabla.

**Conclusiones.**

En esta actividad solo tuve la una complicación al colocar los colores a cada grafica de las partículas, todo lo demas fue muy sencillo siguiendo los pasos del video proporcionado por el profesor.

**Referencias**.

Michel Davalos Boites.[ MICHEL DAVALOS BOITES](04/10/2022) PySide2 – Qscene(Qt for Python)(VI)[Archivo de video]. <https://www.youtube.com/watch?v=3jHTFzPpZY8&ab_channel=MICHELDAVALOSBOITES>

**Código.**

**main.py**

from PySide2.QtWidgets import QPushButton, QApplication

from mainwindow import MainWindow

import sys

#Aplicación de QT

app = QApplication()

#Crear objeto

window = MainWindow()

#Hacer visible el elemento Botón

window.show()

#Qt loop

sys.exit(app.exec\_())

**mainwindow.py**

from ast import Str

from math import fabs

from multiprocessing import managers

from sqlite3 import Row

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 manager import Manager

from particula import Particula

from random import randint

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super(MainWindow, self).\_\_init\_\_()

        self.manager = Manager()

        self.ui = Ui\_MainWindow()

        self.ui.setupUi(self)

        self.id = 0

        #Cuando el botón pushbutton es presionado, ejecuta la función click\_agregar

        # self.ui.mostrar.clicked.connect(self.click\_mostrar)

        self.ui.insertar\_inicio.clicked.connect(self.click\_insertar\_inicio)

        self.ui.insertar\_final.clicked.connect(self.click\_insertar\_final)

        self.ui.mostrar.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\_titulo)

        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(0.8, 0.8)

    @Slot()

    def dibujar(self):

        pen = QPen()

        pen.setWidth(2)

        for particula in self.manager:

            origen\_x = randint (0, 255)

            origen\_y = randint (0, 255)

            destino\_x = randint(0,255)

            destino\_y = randint(0,255)

            color = QColor(origen\_x, origen\_y, destino\_x, destino\_y)

            pen.setColor(color)

            self.scene.addEllipse(origen\_x, origen\_y, 3, 3, pen)

            self.scene.addEllipse(destino\_x, destino\_y, 3, 3,  pen)

            self.scene.addLine(origen\_x+3, origen\_y+3, destino\_x, destino\_y,  pen)

    @Slot()

    def limpiar(self):

        self.scene.clear()

    @Slot()

    def buscar\_titulo(self):

        id = self.ui.buscar\_lineEdit.text() #str

        encontrado = False

        for particula in self.manager:

            if str (id) == str(particula.id):

                self.ui.tabla.clear()

                self.ui.tabla.setRowCount(1)

                id\_widget = QTableWidgetItem(str(particula.id))

                origen\_x\_widget = QTableWidgetItem(str(particula.origen\_x))

                origen\_y\_widget = QTableWidgetItem(str(particula.origen\_y))

                destino\_x\_widget = QTableWidgetItem(str(particula.destino\_x))

                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.tabla.setItem(0, 0, id\_widget)

                self.ui.tabla.setItem(0, 1, origen\_x\_widget)

                self.ui.tabla.setItem(0, 2, origen\_y\_widget)

                self.ui.tabla.setItem(0, 3, destino\_x\_widget)

                self.ui.tabla.setItem(0, 4, destino\_y\_widget)

                self.ui.tabla.setItem(0, 5, velocidad\_widget)

                self.ui.tabla.setItem(0, 6, red\_widget)

                self.ui.tabla.setItem(0, 7, green\_widget)

                self.ui.tabla.setItem(0, 8, blue\_widget)

                self.ui.tabla.setItem(0, 9, distancia\_widget)

                encontrado = True

                return

        if not encontrado:

            QMessageBox.warning(

                self,

                "Atencion",

                f'La particula con nombre "{id}" no fue encontrado'

                )

    @Slot()

    def mostrar\_tabla(self):

        self.ui.tabla.setColumnCount(10)

        headers = ["ID", "Origen X", "Origen Y", "Destino X",

                   "Destino Y", "Velocidad", "Red", "Green", "Blue", "Distancia"]

        self.ui.tabla.setHorizontalHeaderLabels(headers)

        self.ui.tabla.setRowCount(len(self.manager))

        row = 0

        for particula in self.manager:

            id\_widget = QTableWidgetItem(str(particula.id))

            origen\_x\_widget = QTableWidgetItem(str(particula.origen\_x))

            origen\_y\_widget = QTableWidgetItem(str(particula.origen\_y))

            destino\_x\_widget = QTableWidgetItem(str(particula.destino\_x))

            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.tabla.setItem(row, 0, id\_widget)

            self.ui.tabla.setItem(row, 1, origen\_x\_widget)

            self.ui.tabla.setItem(row, 2, origen\_y\_widget)

            self.ui.tabla.setItem(row, 3, destino\_x\_widget)

            self.ui.tabla.setItem(row, 4, destino\_y\_widget)

            self.ui.tabla.setItem(row, 5, velocidad\_widget)

            self.ui.tabla.setItem(row, 6, red\_widget)

            self.ui.tabla.setItem(row, 7, green\_widget)

            self.ui.tabla.setItem(row, 8, blue\_widget)

            self.ui.tabla.setItem(row, 9, distancia\_widget)

            row += 1

    #Funcion que es llamada por x razón que imprime Click en Terminal.

    @Slot()

    # def click\_mostrar(self):

    #     a

    @Slot()

    def action\_abrir\_archivo(self):

        #print("Abrir\_archivo")

        ubicacion = QFileDialog.getOpenFileName(

            self,

            'Abrir Archivo',

            '.',

            'JSON (\*.json)'

        ) [0]

        if self.manager.abrir(ubicacion):

            QMessageBox.information(

                self,

                "Éxito",

                "Se abrió el archivo" + ubicacion

            )

        else:

            QMessageBox.critical(

                self,

                "Error",

                "Error al abrir el archivo" + ubicacion

            )

    @Slot()

    def action\_guardar\_archivo(self):

        #print("guardar\_archivo")

        ubicacion = QFileDialog.getSaveFileName(

            self,

            'Guardar Archivo',

            '.',

            'JSON (\*.json)'

        )[0]

        print(ubicacion)

        if self.manager.guardar(ubicacion):

            QMessageBox.information(

                self,

                "Exito",

                "Se pudo crear el archivi" + ubicacion

            )

        else:

            QMessageBox.critical(

                self,

                "Error",

                "No se pudo crear el archivo" + ubicacion

            )

    def click\_insertar\_inicio(self):

        self.id += 1

        aux = Particula(self.id, self.ui.ox.value(), self.ui.oy.value(), self.ui.dx.value(), self.ui.dy.value(), self.ui.velocidad.value(), self.ui.red.value(), self.ui.green.value(), self.ui.blue.value())

        self.manager.agregarInicio(aux)

        self.click\_mostrar()

    def click\_insertar\_final(self):

        self.id += 1

        aux = Particula(self.id , self.ui.ox.value(), self.ui.oy.value(), self.ui.dx.value(), self.ui.dy.value(), self.ui.velocidad.value(), self.ui.red.value(), self.ui.green.value(), self.ui.blue.value())

        self.manager.agregarFinal(aux)

        self.click\_mostrar()

    def click\_mostrar(self):

        self.ui.lista\_particulas.clear()

        self.ui.lista\_particulas.insertPlainText(str(self.manager))