

Actividad 08 - QScene

**JAIRO CAIN SANCHEZ
ESTRADA// Luis Angel
Elisea Graciano**

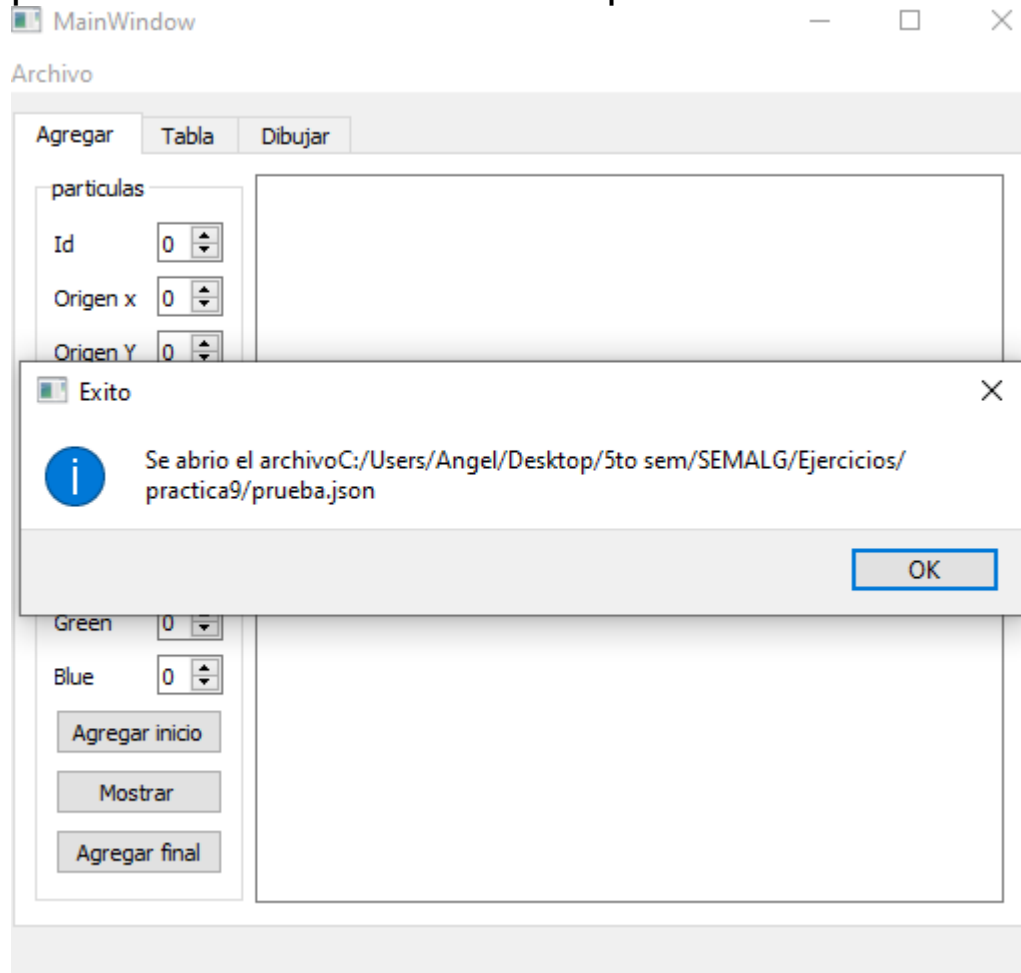
**SEMINARIO DE SOLUCION DE PROBLEMAS DE
ALGORITMIA**

Lineamientos de evaluación

- Se agrego en la interfaz el Qscene para poder mostrar las partículas dibujadas
- Se muestran las partículas que estén alojadas en nuestra clase que las administra sin problema alguno

Desarrollo

Se implemento en la interfaz un QScene para mostrar las particulas alojadas en nuestra clase que las administra, para demostrarlo. Recuperamos las partículas desde nuestro archivo JSON y posteriormente mostramos las partículas existentes.



MainWindow

Archivo

Agregar

Tabla

Dibujar

	Id	Origen_x	Origen_y	Destino_x	Destino_y	Velocidad	Red
1	6	30	11	60	61	61	5
2	10	23	20	60	90	90	5
3	6	50	99	55	55	55	5
4	8	80	4	4	2	2	5
5	66	66	2	88	88	88	5
6	52	88	88	5	5	5	5
7	11	5	5	66	66	66	5
8	5	77	77	22	22	22	5
9	4	65	66	88	88	88	5
10	44	99	55	0	52	52	5
11	47	0	0	99	99	99	5

<

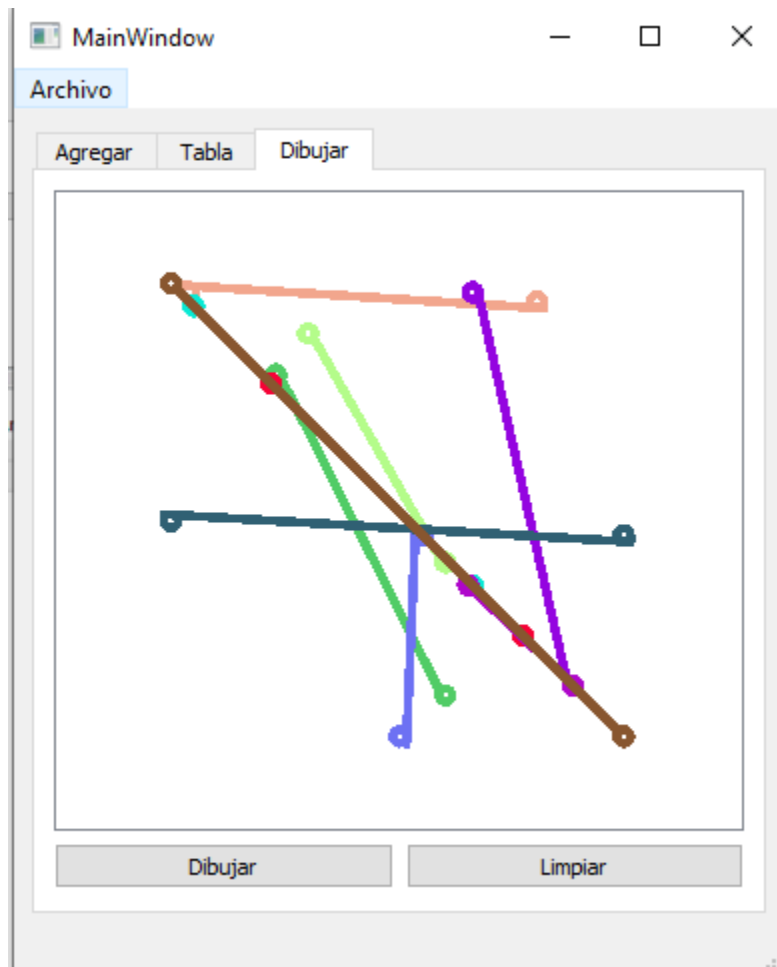
>

Tipo de partícula

Buscar

Mostrar

Posteriormente dibujamos las partículas.



Conclusiones

En esta practica se me hizo interesante como implementamos estos widgets y me interesa saber cómo se seguirán desarrollando las siguientes prácticas.

Referencias

PySide2 - QScene (Qt for Python)(VI). (2020, November 5).
<https://youtu.be/3jHTFzPpZY8>

Código

Particula.py

```
from algoritmos import distancia_euclidiana
class Particula:
    def
__init__(self,id=0,origen_x=0,origen_y=0,destino_x=0,destino_y=0,velocidad=0,red=0,
green=0,blue=0,distancia=0.0):
    self.__id=id
    self.__origen_x=origen_x
    self.__origen_y=origen_y
    self.__destino_x=destino_x
    self.__destino_y=destino_y
    self.__velocidad=velocidad
    self.__red=red
    self.__green=green
    self.__blue=blue
    self.__distancia=distancia_euclidiana(origen_x, destino_x, origen_y,
destino_y)

    def __str__(self):
        return(
            'Id: ' + str(self.__id) + '\n' +
            'Origen x: ' + str(self.__origen_x) + '\n' +
            'Origen y: ' + str(self.__origen_y) + '\n' +
            'Destino x: ' + str(self.__destino_x) + '\n' +
            'Destino y: ' + str(self.__destino_y) + '\n' +
            'Velocidad: ' + str(self.__velocidad) + '\n' +
            'Red: ' + str(self.__red) + '\n' +
            'Green: ' + str(self.__green) + '\n' +
            'Blue: ' + str(self.__blue) + '\n' +
            'Distancia: ' + str(self.__distancia) + '\n'
        )
    @property
    def id(self):
        return self.__id
    @property
    def origen_x(self):
        return self.__origen_x
    @property
    def origen_y(self):
        return self.__origen_y
    @property
    def destino_x(self):
```

```

        return self.__destino_x
@property
def destino_y(self):
    return self.__destino_y
@property
def velocidad(self):
    return self.__velocidad
@property
def red(self):
    return self.__red
@property
def green(self):
    return self.__green
@property
def blue(self):
    return self.__blue
@property
def distancia(self):
    return self.__distancia

def to_dic(self):
    return{
        'id':self.__id,
        'origen_x':self.__origen_x,
        'origen_y':self.__origen_y,
        'destino_x':self.__destino_x,
        'destino_y':self.__destino_y,
        'velocidad':self.__velocidad,
        'red':self.__red,
        'green':self.__green,
        'blue':self.__blue,
        #'distancia':self.__distancia

    }

```

Algoritmos.py

```
from math import sqrt
def distancia_euclidiana(x_1, y_1, x_2, y_2):
    """ Calcula la distancia euclidiana
    Devuelve el resultado de la fórmula
    También se le conoce a la fórmula como:
    distancia entre dos puntos
    Parámetros:
    x_1 -- origen_x
    y_1 -- origen_y
    x_2 -- destino_x
    y_2 -- destino_y
    """
    return(sqrt(((x_2 - x_1)**2) + ((y_2 - y_1)**2)))
```

particulas.py

```
from particula import Particula
import json
class Particulas:
    def __init__(self):
        self.__particulas = []

    def agregar_inicio(self,particula:Particula):
        self.__particulas.insert(0,particula)

    def agregar_final(self,particula:Particula):
        self.__particulas.append(particula)

    def mostrar(self):
        for particula in self.__particulas:
            print(particula)

    def __str__(self):
        return "".join(
            str(particula) + '\n' for particula in self.__particulas
        )

    def __len__(self):
        return len(self.__particulas)

    def __iter__(self):
        self.cont=0
        return self
```

```

def __next__(self):
    if self.cont < len(self.__particulas):
        particula =self.__particulas[self.cont]
        self.cont+=1
        return particula
    else:
        raise StopIteration

def guardar(self,ubicacion):
    try:
        with open(ubicacion,"w") as archivo:
            lista=[particula.to_dic() for particula in self.__particulas]
            print(lista)
            json.dump(lista,archivo, indent=5)
        return 1
    except:
        return 0

def abrir(self,ubicacion):
    try:
        with open(ubicacion,"r") as archivo:
            lista=json.load(archivo)
            self.__particulas=[Particula(**particula) for particula in lista]
        return 1
    except:
        return 0

```

mainwindow.py

```

from PySide2.QtWidgets import QMainWindow, QFileDialog,
QMessageBox,QTableWidgetItem,QGraphicsScene
from PySide2.QtCore import Slot
from ui_mainwindow import Ui_MainWindow
from particula import Particula
from particulas import Particulas
from PySide2.QtGui import QPen,QColor,QTransform
from random import randint

```

```

#pyside2-uic mainwindow.ui para pasar de .ui a python

```



```

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()

        self.particulas= Particulas()

        self.ui=Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.Agregar_final_pushButton.clicked.connect(self.click_agregar)
        self.ui.Agregar_inicio_pushButton.clicked.connect(self.click_agregar_inicio
)

        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_id)

        self.ui.dibujar_pushButton.clicked.connect(self.dibujar)
        self.ui.limpiar_pushButton.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 partícula in self.particulas:

            origen_x=partícula.origen_x
            origen_y=partícula.origen_y
            destino_x=partícula.destino_x
            destino_y=partícula.destino_y
            velocidad= partícula.destino_y
        #for i in range(200):
            r=randint(0,255)
            g=randint(0,255)
            b=randint(0,255)
            color=QColor(r,g,b)

```

```

        pen.setColor(color)

        #origen_x=randint(0,500)
        #origen_y=randint(0,500)
        #destino_x=randint(0,500)
        #destino_y=randint(0,500)

        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_id(self):
        id=self.ui.buscar_lineEdit.text()
        encontrado=False
        for partícula in self.partículas:
            if id==partícula.id:
                self.ui.tabla.clear()
                self.ui.tabla.setRowCount(1)

                id_widget= QTableWidgetItem(str(partícula.id))
                origen_x_widget= QTableWidgetItem(str(partícula.origen_x))
                origen_y_widget= QTableWidgetItem(str(partícula.origen_y))
                destino_x_widget= QTableWidgetItem(str(partícula.destino_x))
                destino_y_widget= QTableWidgetItem(str(partícula.destino_y))
                velocidad_widget= QTableWidgetItem(str(partícula.destino_y))
                red_widget= QTableWidgetItem(str(partícula.red))
                green_widget= QTableWidgetItem(str(partícula.green))
                blue_widget= QTableWidgetItem(str(partícula.blue))
                distancia_widget= QTableWidgetItem(str(partícula.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 partícula "{id}" no fue encontrada'
            )

    @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.particulas))
        row=0
        for partícula in self.particulas:
            id_widget= QTableWidgetItem(str(partícula.id))
            origen_x_widget= QTableWidgetItem(str(partícula.origen_x))
            origen_y_widget= QTableWidgetItem(str(partícula.origen_y))
            destino_x_widget= QTableWidgetItem(str(partícula.destino_x))
            destino_y_widget= QTableWidgetItem(str(partícula.destino_y))
            velocidad_widget= QTableWidgetItem(str(partícula.destino_y))
            red_widget= QTableWidgetItem(str(partícula.red))
            green_widget= QTableWidgetItem(str(partícula.green))
            blue_widget= QTableWidgetItem(str(partícula.blue))
            distancia_widget= QTableWidgetItem(str(partícula.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

    @Slot()
    def action_abrir_archivo(self):
        #print("abrir")
        ubicacion=QFileDialog.getOpenFileName(
            self,
            "Abrir Archivo",
            ".",

```

```

        "JSON (*.json)"
    )[0]
    if self.particulas.abrir(ubicacion):
        QMessageBox.information(
            self,
            "Exito",
            "Se abrio el archivo" + ubicacion
        )
    else:
        QMessageBox.critical(
            self,
            "Error",
            "No se pudo abrir el archivo"
        )
@Slot()
def action_guardar_archivo(self):
    #print("guardar")
    ubicacion=QFileDialog.getSaveFileName(
        self,
        "Guardar Archivo",
        ".",
        "JSON (*.json)"
    )[0]
    print(ubicacion)
    if self.particulas.guardar(ubicacion):
        QMessageBox.information(
            self,
            "Exito",
            "Se pudo crear el archivo" + ubicacion
        )
    else:
        QMessageBox.critical(
            self,
            "Error",
            "No se pudo crear el archivo"
        )
@Slot()
def click_mostrar(self):
    self.ui.salida.clear()
    self.ui.salida.insertPlainText(str(self.particulas))

@Slot()
def click_agregar(self):
    id=self.ui.Id_spinBox.text()
    origen_x=self.ui.Origen_x_spinBox.value()

```

```

        origen_y=self.ui.Origen_y_spinBox.value()
        destino_x=self.ui.Destino_x_spinBox.value()
        destino_y=self.ui.Destino_y_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()

        Particula1=Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,red
,green,blue)
        self.particulas.agregar_final(Particula1)

    @Slot()
    def click_agregar_inicio(self):
        id=self.ui.Id_spinBox.text()
        origen_x=self.ui.Origen_x_spinBox.value()
        origen_y=self.ui.Origen_y_spinBox.value()
        destino_x=self.ui.Destino_x_spinBox.value()
        destino_y=self.ui.Destino_y_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()

        Particula1=Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,red
,green,blue)
        self.particulas.agregar_inicio(Particula1)

```

Main.py

```

from PySide2.QtWidgets import QApplication
from mainwindow import MainWindow
#pyside2-uic mainwindow.ui para pasar de .ui a python
#pyside2-uic mainwindow.ui >ui_mainwindow.py
import sys
app=QApplication()
window=MainWindow()
window.show()
sys.exit(app.exec_())

```