

Actividad 9 (QScene)

Beracoechea Rosales Jose Francisco

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 las partículas del antes y después de ser ordenadas por id de manera ascendente tanto en el `QPlainTextEdit` como en el `QTableWidget`.
- ☐ Se muestra captura de pantalla de las partículas del antes y después de ser ordenadas por distancia de manera descendente tanto en el `QPlainTextEdit` como en el `QTableWidget`.
- ☐ Se muestra captura de pantalla de las partículas del antes y después de ser ordenadas por velocidad de manera ascendente tanto en el `QPlainTextEdit` como en el `QTableWidget`.

DESARROLLO:

Sin ordenar:

Agregar

Tabla

Grafica

PARTICULAS

ID

13

ORIGEN X

18

ORIGEN Y

32

DESTINO X

8

DESTINO Y

31

VELOCIDAD

136

ROJO

0

VERDE

0

AZUL

0

AGREGAR AL FINAL

AGREGAR AL INICIO

MOSTRAR

id = 7

origen_x = 15

origen_y = 16

destino_x = 8

destino_y = 31

velocidad = 600

red = 0

green = 0

blue = 0

distancia = 16.55294535724685

id = 2

origen_x = 7

origen_y = 7

destino_x = 6

destino_y = 6

velocidad = 6

red = 0

green = 0

blue = 0

distancia = 1.4142135623730951

id = 1

origen_x = 5

origen_y = 5

destino_x = 3

destino_y = 3

velocidad = 2

red = 0

green = 0

blue = 0

distancia = 2.8284271247461903

Agregar

Tabla

Grafica

	id	origen_x	origen_y	destino_x
3	3	19	17	28
2	2	17	15	25
1	1	15	10	19
4	4	45	23	2
15	15	99	23	2

Buscar

Mostrar

Ordenar por velocidad

Ordenar por distancia

Ordenar por ID

Ordenado por ID:

Agregar

Tabla

Grafica

	id	origen_x	origen_y	destino_x
1	1	15	10	19
2	2	17	15	25
3	3	19	17	28
4	4	45	23	2
5	15	99	23	2

ID

Buscar

Mostrar

Ordenar por velocidad

Ordenar por distancia

Ordenar por ID

Agregar

Tabla

Grafica

PARTICULAS

ID

13

ORIGEN X

18

ORIGEN Y

32

DESTINO X

8

DESTINO Y

31

VELOCIDAD

136

ROJO

0

VERDE

0

AZUL

0

AGREGAR AL FINAL

AGREGAR AL INICIO

MOSTRAR

id = 1

origen_x = 5

origen_y = 5

destino_x = 3

destino_y = 3

velocidad = 2

red = 0

green = 0

blue = 0

distancia = 2.8284271247461903

id = 2

origen_x = 7

origen_y = 7

destino_x = 6

destino_y = 6

velocidad = 6

red = 0

green = 0

blue = 0

distancia = 1.4142135623730951

id = 3

origen_x = 10

origen_y = 11

destino_x = 7

destino_y = 7

velocidad = 6

red = 0

green = 0

blue = 0

distancia = 1.4142135623730951


```

Codigo: from PySide2.QtWidgets import
QApplication from mainwindow import
MainWindow import sys app =
QApplication()
    window =
MainWindow()

window.show()

sys.exit(app.exec_())

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 particulas import Particula from Lista
import Lista
    class
MainWindow(QMainWindow):
def __init__(self):
    super(MainWindow, self).__init__()          self.lista =
Lista()          self.ui = Ui_MainWindow()
self.ui.setupUi(self)
self.ui.inicio_pushButton.clicked.connect(self.click_agregar)
self.ui.FINAL_pushButton.clicked.connect(self.click_final)
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.orden_dist.clicked.connect(self.orden_d)
self.ui.orden_id.clicked.connect(self.orden_id)
self.ui.orden_vel.clicked.connect(self.orden_vel)

        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(3)
        for Particula in
self.lista:
            r = Particula.red
b = Particula.blue          g
= Particula.green

```

```

color = QColor(r,b,g)
pen.setColor(color)

self.scene.addEllipse(Particula.origen_x,Particula.origen_y,3,3,pen)

self.scene.addEllipse(Particula.destino_x,Particula.destino_y,3,3,pen)

self.scene.addLine(3+Particula.origen_x,3+Particula.origen_y,Particula.destino_x,Particula.destino_y,pen)

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

    @Slot()    def
orden_d(self):
    self.lista.ordenar_distancia()
self.mostrar_tabla()

    @Slot()    def
orden_id(self):
    self.lista.ordenar_id()
self.mostrar_tabla()
    @Slot()
def orden_vel(self):
    self.lista.ordenar_velocidad()
self.mostrar_tabla()

    @Slot()    def buscar_id(self):
id = self.ui.Buscar_lineEdit.text()
encontrado = False    for Particula in
self.lista:    if id ==
(str(Particula.id)):
        self.ui.tableWidget.clear()
self.ui.tableWidget.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))
        blue_widget = QTableWidgetItem(str(Particula.blue))
        green_widget = QTableWidgetItem(str(Particula.green))
        distancia_widget =
QTableWidgetItem(str(Particula.distancia))

        self.ui.tableWidget.setItem(0,0,id_widget)
self.ui.tableWidget.setItem(0,1,origen_x_widget)
self.ui.tableWidget.setItem(0,2,origen_y_widget)
self.ui.tableWidget.setItem(0,3,destino_x_widget)

```

```

self.ui.tableWidget.setItem(0,4,destino_y_widget)
self.ui.tableWidget.setItem(0,5,velocidad_widget)
self.ui.tableWidget.setItem(0,6,red_widget)
self.ui.tableWidget.setItem(0,7,blue_widget)
self.ui.tableWidget.setItem(0,8,green_widget)
self.ui.tableWidget.setItem(0,9,distancia_widget)
        encontrado = True
return         if not encontrado:
QMessageBox.warning(         self,
"ATENCION",         f'LA PARTICULA"{id}"NO
FUE ENCONTADA'
        )

@Slot()         def
mostrar_tabla(self):
        self.ui.tableWidget.setColumnCount(10)
headers = ["id","origen_x",
"origen_y","destino_x","destino_y","velocidad","red","green","blue","distan
cia"]         self.ui.tableWidget.setHorizontalHeaderLabels(headers)

self.ui.tableWidget.setRowCount(len(self.lista))

        row = 0         for
Particula in self.lista:
        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))         blue_widget
= QTableWidgetItem(str(Particula.blue))         green_widget =
QTableWidgetItem(str(Particula.green))         distancia_widget =
QTableWidgetItem(str(Particula.distancia))

self.ui.tableWidget.setItem(row,0,id_widget)
self.ui.tableWidget.setItem(row,1,origen_x_widget)
self.ui.tableWidget.setItem(row,2,origen_y_widget)
self.ui.tableWidget.setItem(row,3,destino_x_widget)
self.ui.tableWidget.setItem(row,4,destino_y_widget)
self.ui.tableWidget.setItem(row,5,velocidad_widget)
self.ui.tableWidget.setItem(row,6,red_widget)
self.ui.tableWidget.setItem(row,7,blue_widget)
self.ui.tableWidget.setItem(row,8,green_widget)
self.ui.tableWidget.setItem(row,9,distancia_widget)

        row += 1

@Slot()         def
action_abrir_archivo(self):
        ubicacion =
QFileDialog.getOpenFileName(         self,
        'Abrir archivo',
        '.',

```

```

        'JSON (*.json) '
    )[0]
    if
self.lista.abrir(ubicacion):
QMessageBox.information(
self,
        "EXITO",
        "SE ABRIO CON EXITO EL ARCHIVO" + ubicacion
    )
else:

QMessageBox.critical(
self,
        "ERORR",
        "ERROR AL ABRIR EL ARCHIVO" + ubicacion
    )

```

```

@Slot()
def
action_guardar_archivo(self):
    ubicacion = QFileDialog.getSaveFileName(
self,
        'Guardar',
        '.',
        'JSON (*.json) '
    )[0]
    print(ubicacion)
if self.lista.guardar(ubicacion):
QMessageBox.information(
self,
        "EXITO",
        "SE PUDO CREAR EL ARCHIVO" + ubicacion,
    )
else :
    QMessageBox.critical(
        self,
        "ERROR",
        "NO SE PUDO CREAR EL ARCHIVO" + ubicacion
    )

```

```

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

```

```

@Slot()
def
click_agregar(self):
    id = self.ui.ID_spinBox.value()
    origen_x = self.ui.ORIGEN_X_spinBox.value()
    origen_y = self.ui.ORIGEN_Y_spinBox.value()
    destino_x = self.ui.x_spinBox.value()
    destino_y = self.ui.y_spinBox.value()
    velocidad = self.ui.velocidad_spinBox.value()
    rojo = self.ui.rojo_spinBox.value()
    verde = self.ui.verde_spinBox.value()
    azul = self.ui.azul_spinBox.value()
    partícula
=

```

```

Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,rojo,verde,azu
l)
    self.lista.agregar_inicio(particula)

    @Slot()
    def
click_final(self):
    id = self.ui.ID_spinBox.value()
    origen_x = self.ui.ORIGEN_X_spinBox.value()
    origen_y = self.ui.ORIGEN_Y_spinBox.value()
    destino_x = self.ui.x_spinBox.value()
    destino_y = self.ui.y_spinBox.value()
    velocidad = self.ui.velocidad_spinBox.value()
    rojo = self.ui.rojo_spinBox.value()
    verde = self.ui.verde_spinBox.value()
    azul = self.ui.azul_spinBox.value()

    particula =
Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,rojo,verde,azu
l)
    self.lista.agregar_final(particula)
    from particulas import
Particula
    import json
    class
Lista:
    def __init__
(self):
        self.__Lista = []
    def agregar_final(self, particulas:Particula
):
        self.__Lista.append(particulas)
    def agregar_inicio(self,
particulas:Particula ):
        self.__Lista.insert(0, particulas)

    def mostrar(self):
        for
particulas in self.__Lista:
            print(particulas)

    def
ordenar_id(self):
        self.__Lista.sort()

    def
ordenar_velocidad(self):
        self.__Lista.sort(key = Particula.__sort_by_velocidad__)

    def
ordenar_distancia(self):
        self.__Lista.sort(key = Particula.__sort_by_distancia__,reverse=
True)

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

)
    def guardar (self,
ubicacion):
        try:
            with open(ubicacion, 'w') as
archivo:
                lista = [particulas.to_dict() for particulas in
self.__Lista]

```



```

        json.dump(lista,
archivo, indent=11)
        return 1
except:
    return 0
    def abrir(self,
ubicacion):
        try:
            with open(ubicacion, 'r') as archivo:
                lista = json.load(archivo)
            = [Particula(**particulas) for particulas in lista]
        except:
            return 0
def __len__(self):
    return len(self.__Lista)
    def
__iter__(self):
self.cont = 0
return self
    def __next__(self):
        if self.cont <
len(self.__Lista):
            Particula =
self.__Lista[self.cont]
            self.cont
+= 1
            return Particula
        else
:
            raise StopIteration
## particulas.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):
        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,
origen_y, destino_x, 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'
    )
    def
to_dict(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,
    }

    def
__lt__(self,Particula)->bool:
return self.id < Particula.id
    def
__sort_by_velocidad__(Particula):
return Particula.__velocidad
    def
__sort_by_distancia__(Particula):
return Particula.__distancia

```

```

    @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
blue(self):        return
self.__blue

    @property    def
green(self):        return
self.__green

    @property    def
distancia(self):    return
self.__distancia

```

CONCLUSIONES:

En esta actividad aprendí a ordenar elementos de una lista de 3 formas diferentes aunque se reduce a 2 formas de programarla ya que no podemos ordenar por más de una característica al mismo tiempo

Referencias: <https://youtu.be/3jHTFzPpZY8>