# Actividad 9 (QScene)

Beracoechea Rosales Jose Francisco

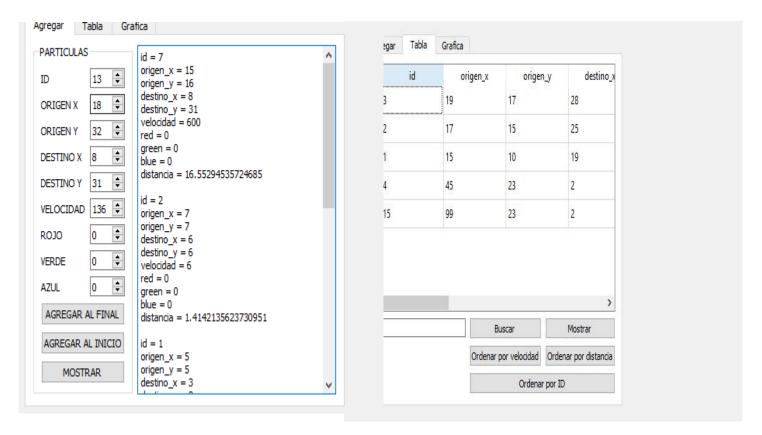
Seminario de algoritmia

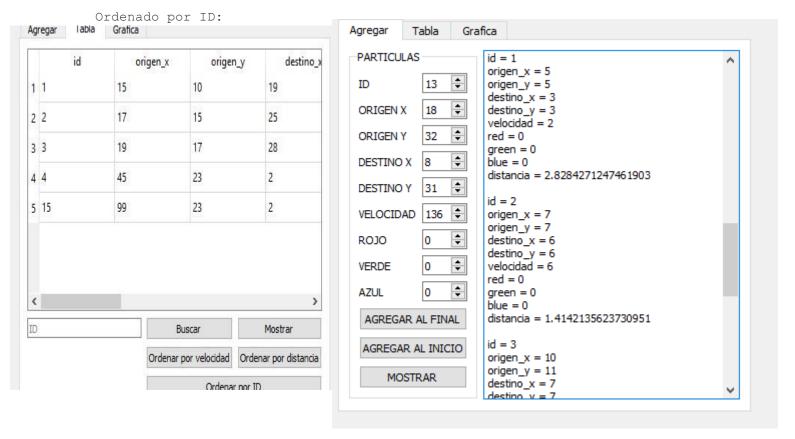
# Lineamientos de evaluación:

- [] 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 <u>Formato de Actividades</u>.
- [] Se muestra captura de pantalla de las partículas del antes y después de ser ordenadas por id de manera ascendente tanto en el <code>QPlainTextEdit</code> como en el <code>QTableWidget</code>.
- [] 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.

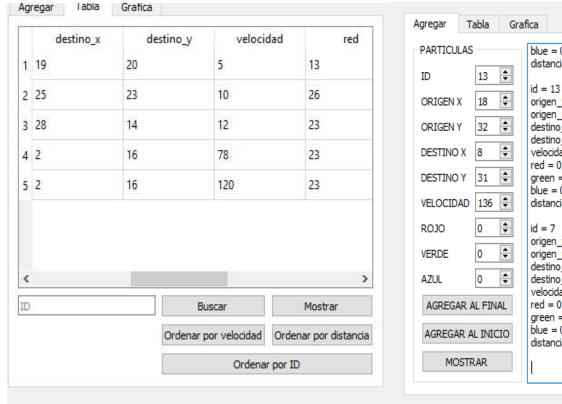
#### **DESARROLLO:**

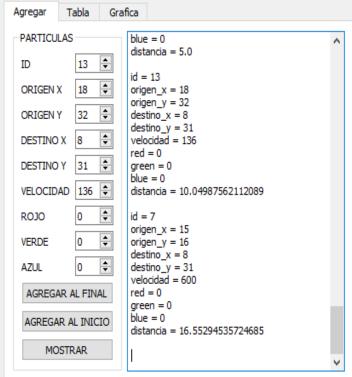
### Sin ordenar:



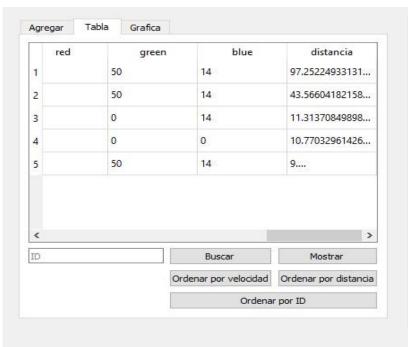


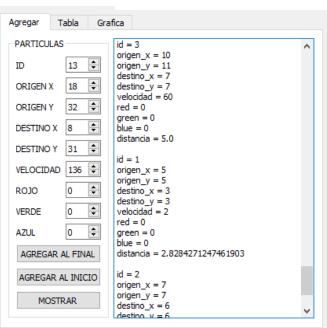
#### Ordenado por velocidad:





#### Ordenado por distancia descendente:





```
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 =
              self.ui = Ui MainWindow()
Lista()
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,
               if event.delta()
event):
> 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
= Particula.green
```

Codigo: from PySide2.QtWidgets import

```
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.dest
ino 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
                       if id ==
self.lista:
(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
              self.ui.tableWidget.setHorizontalHeaderLabels(headers)
self.ui.tableWidget.setRowCount(len(self.lista))
        row = 0
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',
        1.1,
```

```
'JSON (*.json)'
                  if
       ) [0]
self.lista.abrir(ubicacion):
QMessageBox.information(
                  "EXITO",
self,
            "SE ABRIO CON EXITO EL ARCHIVO" + ubicacion
        )
else:
QMessageBox.critical(
                      "ERORR",
self,
                "ERROR AL ABRIR EL ARCHIVO" + ubicacion
        )
     @Slot() def
action guardar archivo(self):
     ubicacion = QFileDialog.getSaveFileName(
self,
            'Guardar',
            1.1,
            'JSON (*.json)'
                  print(ubicacion)
if self.lista.guardar(ubicacion):
QMessageBox.information(
                  "EXITO",
self,
            "SE PUDO CREAR EL ARCHIVO" + ubicacion,
else :
        QMessageBox.critical(
           self,
"ERROR",
           "NO SE PUDO CREAR EL ARCHIVO" + ubicacion
        )
    @Slot()
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()
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
          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()
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
          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)
                                   for
     def mostrar(self):
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=
      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:
                                                         self. Lista
              lista = json.load(archivo)
= [Particula(**particulas) for particulas in lista]
                                                            return 1
except:
          return 0
def len (self):
       return len(self. Lista)
              def
 iter (self):
self.cont = 0
return self
   def __next__(self):
                              if self.cont <</pre>
len(self. Lista):
                                Particula =
self. Lista[self.cont]
                                      self.cont
                    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.__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
__sort_by_velocidad__(Particula):
return Particula. velocidad
__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
                  def
     @property
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: <a href="https://youtu.be/3jHTFzPpZY8">https://youtu.be/3jHTFzPpZY8</a>