Actividad 11 - Fuerza bruta

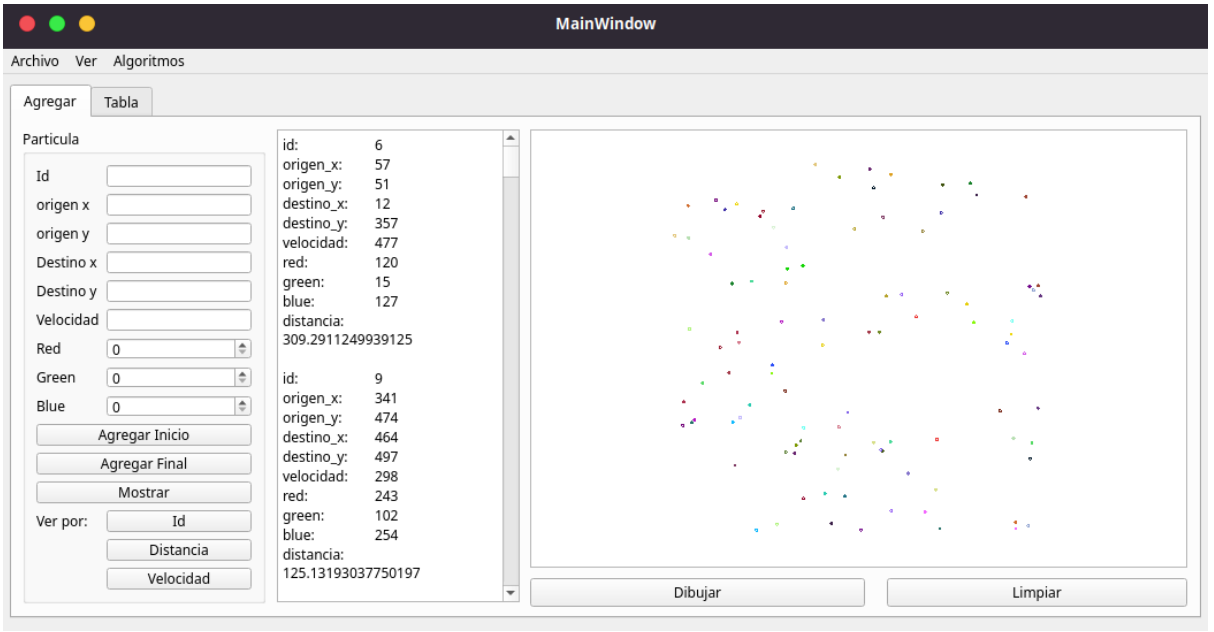
Gabriel Eduardo Sevilla Chavez

Seminario de algoritmia

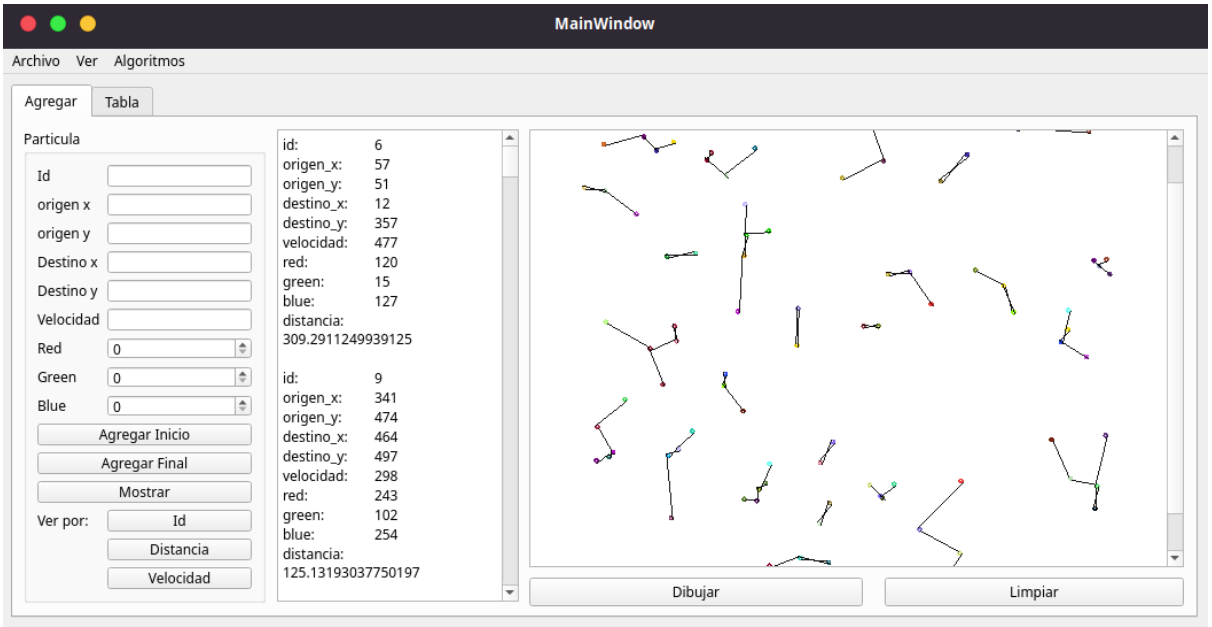
Lineamientos de evaluación

- [ ] 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 los puntos de las partículas en el `QScene`.
- [ ] Se muestra captura de pantalla del resultado del algoritmo de fuerza bruta en el `QScene`.

Puntos de las partículas en el `QScene`.



Algoritmo de fuerza bruta en el `QScene`.



Conclusiones

Fue un poco difícil adaptar el algoritmo con las partículas, pues sí había que encontrar la manera de realizarlo, conjuntar los datos en listas y usarlos de tal manera que el algoritmo funcionase. Pero al final pudo realizarse en su tiempo en forma mostrando los resultados esperados.

Referencias

MICHEL DAVALOS BOITES. (2021, 19 octubre). *Clase Fuerza Bruta (19.oct.21)*.

YouTube. https://www.youtube.com/watch?v=zmPOdDMTk0Y

Código

mainWindow.py

```
from PySide2.QtWidgets import QMainWindow, QFileDialog, QMessageBox,
QTableWidgetItem, QGraphicsScene

from PySide2.QtGui import QPen,QColor,QTransform

from PySide2.QtCore import Slot
```

```
from ui_mainWindow import Ui_MainWindow

from random import  randint

#incluir clases particlas

from Actividad05_ClasesyObjetos.particulas import Particulas

from Actividad05_ClasesyObjetos.particula import Particula

from Actividad05_ClasesyObjetos.algoritmos import puntoMasCercano

class MainWindow(QMainWindow):

   def __init__(self):

        super(MainWindow, self).__init__() #Contructor de QMainWindow

        #Guardar particulas

        self.particulas = Particulas()

        self.puntos = []

        self.ui = Ui_MainWindow()

        #mandar los datos de self.ui a la ventana

        self.ui.setupUi(self)

        # Eventos en botones


self.ui.pbAgregarInicio.clicked.connect(self.click_agregarInicio)


self.ui.pbAgregaFinal.clicked.connect(self.click_agregarFinal)

        self.ui.pbMostrar.clicked.connect(self.click_mostrar)

        #Ad Archivo
```

```
self.ui.actionAbrir_archivo.triggered.connect(self.actionOpenFile)

self.ui.actionGuardar_archivo.triggered.connect(self.actionSaveFile)
        #Table
        self.ui.btnMostrarTabla.clicked.connect(self.mostrarTabla)
        self.ui.btnBuscar.clicked.connect(self.buscarId)
        #Dibujo
        self.ui.btnDibujar.clicked.connect(self.dibujar)
        self.ui.btnLimpiar.clicked.connect(self.limpiar)
        #Escena
        self.scene = QGraphicsScene()
        self.ui.graphicsView.setScene(self.scene)
        #Ordenamiento
        self.ui.btnOrdenarId.clicked.connect(self.ordenarIds)

self.ui.btnOrdenDistanca.clicked.connect(self.ordenarDistancias)

self.ui.btnOrdenarVelocidad.clicked.connect(self.ordenarVelocidades)
        #Ver puntos
        self.ui.actionPuntos.triggered.connect(self.dibujarPuntos)
        #Puntos cercanos
```

```
self.ui.actionMas_cercano.triggered.connect(self.DibujarpuntosCercanos)

    #Zoom
    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 ordenarIds(self):
        listParticulas = []
        self.ui.salida.clear()
        for particula in self.particulas:
            listParticulas.append(particula)
        listParticulas.sort()
        for particula in listParticulas:
            self.ui.salida.insertPlainText(str(particula)+'\n')


    @Slot()
    def ordenarVelocidades(self):
        listParticulas = []
```

```
        self.ui.salida.clear()
        for particula in self.particulas:
            listParticulas.append(particula)
        listParticulas.sort(key=lambda particula:particula.velocidad)

        for particulaordenada in listParticulas:

self.ui.salida.insertPlainText(str(particulaordenada)+'\n')

    @Slot()
    def ordenarDistancias(self):
        listParticulas = []
        self.ui.salida.clear()
        for particula in self.particulas:
            listParticulas.append(particula)
        listParticulas.sort(key=lambda
particula:particula.distancia,reverse=True)

        for particulaordenada in listParticulas:

self.ui.salida.insertPlainText(str(particulaordenada)+'\n')

    @Slot()
```

```
    def DibujarpuntosCercanos(self):
        pen = QPen()
        pen.setWidth(1)
        for particula in self.particulas:

self.puntos.append((particula.origen_x,particula.origen_y))

self.puntos.append((particula.destino_x,particula.destino_y))
        resultado = puntoMasCercano(self.puntos)

        for punto1,punto2 in resultado:
            x1 = punto1[0]
            y1 = punto1[1]
            x2 = punto2[0]
            y2 = punto2[1]
            self.scene.addLine(x1+3,y1+3,x2,y2,pen)

    @Slot()
    def dibujarPuntos(self):
        pen = QPen()
        pen.setWidth(2)
        for particula in self.particulas:
            r=particula.red
```

```
            g=particula.green
            b=particula.blue
            color = QColor(r,g,b)
            pen.setColor(color)
            origen_x  = particula.origen_x
            origen_y  = particula.origen_y
            destino_x = particula.destino_x
            destino_y = particula.destino_y
            self.scene.addEllipse(origen_x,origen_y,3,3,pen)
            self.scene.addEllipse(destino_x,destino_y,3,3,pen)

@Slot()
def dibujar(self):
    pen = QPen()
    pen.setWidth(2)


    for particula in self.particulas:
        r=particula.red
        g=particula.green
        b=particula.blue
        color = QColor(r,g,b)
```

```
            pen.setColor(color)
            origen_x  = particula.origen_x
            origen_y  = particula.origen_y
            destino_x = particula.destino_x
            destino_y = particula.destino_y
            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 buscarId(self):
        id = self.ui.lineEditTabla.text()
        encontrado = False
        for particula in self.particulas:
            if id == str(particula.id):
                self.ui.tableWidget.clear()
```

```
                self.ui.tableWidget.setRowCount(1)
                self.viewData(0,particula)
                encontrado = True
                return
        if not encontrado:
            QMessageBox.warning(
                self,"Atencion",f'El id "{id}" no se encuentra'
            )



    @Slot()
    def mostrarTabla(self):
        self.ui.tableWidget.setColumnCount(10)
        headers = ["id","origen_x","origen_y","destino_x",

"destino_y","velocidad","red","green","blue","distancia"]
        self.ui.tableWidget.setHorizontalHeaderLabels(headers)
        self.ui.tableWidget.setRowCount(len(self.particulas))
        row = 0
        for particula in self.particulas:
            self.viewData(row,particula)
```

```
            row +=1


    def viewData(self,row,particula):
        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.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,green_widget)
```

```
        self.ui.tableWidget.setItem(row,8,blue_widget)
        self.ui.tableWidget.setItem(row,9,distancia_widget)

    @Slot()
    def actionOpenFile(self):
        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',
                'Error al abrir el archivo'+ubicacion
            )
    def actionSaveFile(self):
```

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

```
    @Slot() #Guardar los datos obenidos
    def click_agregarInicio(self):
        id = self.ui.leId.text()
        origenx = self.ui.leOrigenx.text()
        origeny = self.ui.leOrigenY.text()
        destinox = self.ui.leDestinoX.text()
        destinoy = self.ui.leDestinoY.text()
        velocidad = self.ui.leVelocidad.text()
        red = self.ui.sbRed.value()
        green = self.ui.sbGreen.value()
        blue = self.ui.sbBlue.value()
        #Crear particla
        particula =
Particula(int(id),int(origenx),int(origeny),int(destinox),int(destin
oy),int(velocidad),red,green,blue)
        self.particulas.agregar_inicio(particula)

    @Slot() #Guardar los datos obenidos
    def click_agregarFinal(self):
        id = self.ui.leId.text()
        origenx = self.ui.leOrigenx.text()
        origeny = self.ui.leOrigenY.text()
```

```
        destinox = self.ui.leDestinoX.text()
        destinoy = self.ui.leDestinoY.text()
        velocidad = self.ui.leVelocidad.text()
        red = self.ui.sbRed.value()
        green = self.ui.sbGreen.value()
        blue = self.ui.sbBlue.value()
        #Crear particla
        particulafinal =
Particula(int(id),int(origenx),int(origeny),int(destinox),int(destin
oy),int(velocidad),red,green,blue)
        self.particulas.agregar_final(particulafinal)
```

algoritmos.py

```
import math


def distancia_euclidiana(x1,y1,x2,y2):
    #Formula distancia entre dos puntos
    """Parámetros:
    x1 -- origen_x
    y1 -- origen_y
    x2 -- destino_x
    y2 -- destino_y"""
    result = math.sqrt((x2-x1)**2 + (y2-y1)**2)
```

```
    return result


def puntoMasCercano(puntos:list)->list:
    resultado = []
    for punto_i in puntos:
        x1 = punto_i[0]
        y1 = punto_i[1]
        min = 1000
        cercano = (0,0)
        for punto_j in puntos:
            if punto_i != punto_j:
                x2 = punto_j[0]
                y2 = punto_j[1]
                d = distancia_euclidiana(x1,y1,x2,y2)
                if d < min:
                    min = d
                    cercano = (x2,y2)
        resultado.append((punto_i,cercano))
    return resultado
```