# ACTIVIDAD 11

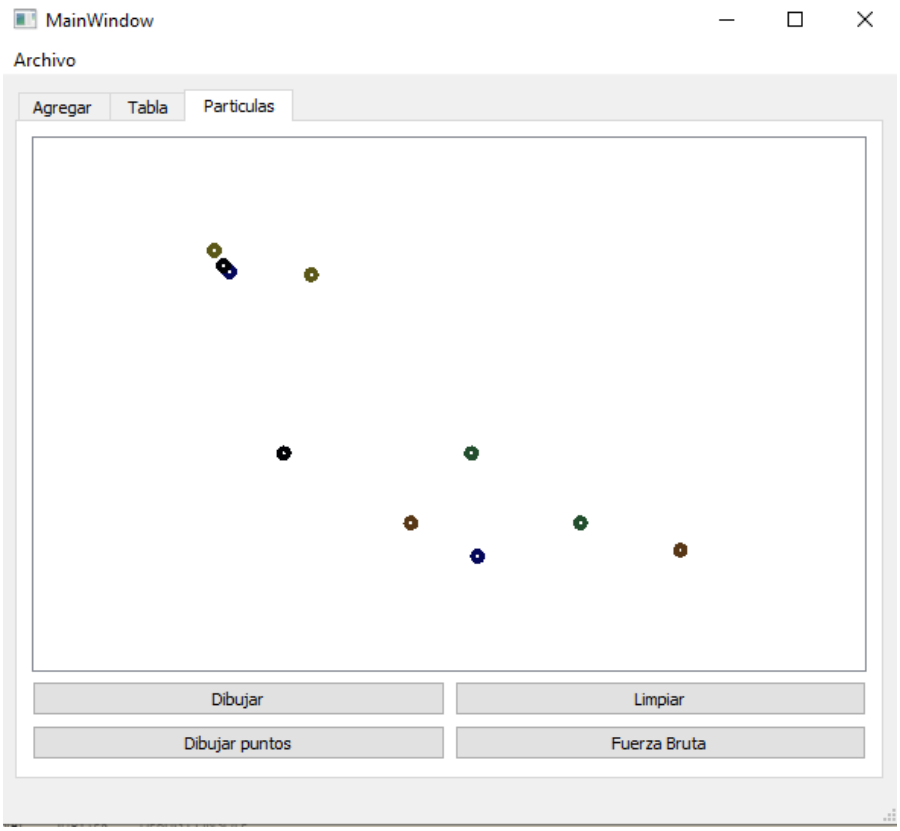
# Fuerza Bruta


## Gomez Casillas Hector Samuel

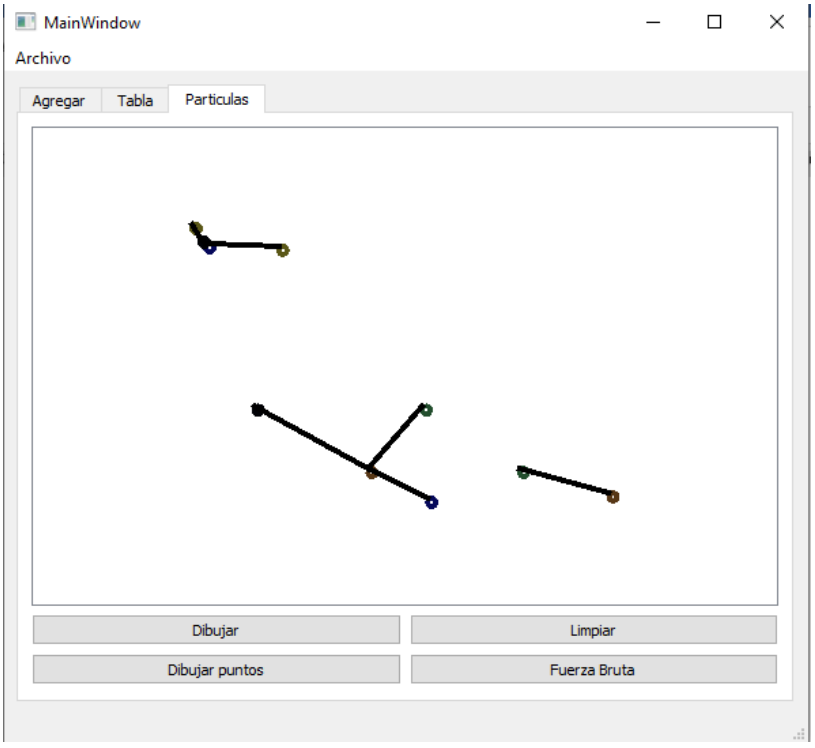

### SEMINARIO DE SOLUCION DE PROBLEMAS DE ALGORITMIA

- El reporte está en formato Goodle 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 particulas en el QScene
- Se muestra captura de pantalla del resultado del algoritmo de fuerza bruta en el QScene

Función que dibuja los puntos de las partículas:




Función que realiza la fuerza bruta:



CONCLUSIONES

Fue una actividad interesante de realizar, tuve que apoyarme de una clase graba del profesor Boites para poder entender a que se referia con método de fuerza bruta, y gracias a ello me ayudó mucho en mi avance, al final solamente fue razonar lo que se necesitaba para aplicar el método de fuerza bruta y lo que nosotros ya teníamos, que era los puntos de las particulas.

REFERENCIAS


- Clase Fuerza Bruta (19.oct.21) (MICHEL DAVALOS BOITES). https://www.youtube.com/watch?v=zmPOdDMTk0Y&t=1969s

Codigo “administradora.py”

```
from particula import Particula
import json
from algoritmos import puntos_mas_cercanos
#Clase que adiministra las particulas
class Administradora:
    def __init__(self):
        self.__particulas = [] #Lista que contendra las particulas y sus
respectivas caracteristicas

    def agregar_final(self,particula:Particula):
        self.__particulas.append(particula) #Agregar al final de la lista
__particulas una particula

    def agregar_inicio(self,particula:Particula):
        self.__particulas.insert(0,particula)#Agregar al inicio de la lista
__particulas una particula

    def mostrar(self): #Imprimir las particulas que haya en la lista
__particulas
        for particula in self.__particulas:
            print(particula)

    def __str__(self):
        return "".join(
            str(particula) 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,ubiacion):
        try:
            with open(ubiacion,'w') as archivo:
                lista = [particula.to_dict() for particula in
self.__particulas]
                json.dump(lista,archivo, indent = 5)
            return
        except:
            return 0
             #json.dump()

    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

    def ordenarID(self):
        return self.__particulas.sort(key=lambda particula: particula.id)


    def ordenarDistancia(self):
        return self.__particulas.sort(key=lambda particula:
particula.distancia)

    ##########################################################################
################################
    def PC(self):
        par = [] #lista para guardar puntos
        par2 = [] #lista para guardar puntos
        for particula in self.__particulas:#por cada particula en la lista
de particulas
            x1 = particula.origen_x#guardo la coordenada en x del primer
punto
            y1 = particula.origen_y#guardo la coordenada en y del primer
punto

            x2 = particula.destino_x#guardo la coordenada en x del segundo
punto
```

```
            y2 = particula.destino_y#guardo la coordenada en y del segundo
punto

            x = (x1,y1)#guardo en la variable x las coordenadas del primer
punto
            y = (x2,y2)#guardo en la variable y las coordenadas del segundo
punto

            par2.append(y)#agrego el segundo punto a la segunda lista
            par.append(x)#agrego el primer punto a la segunda lista

        lista = par2 + par   #junto las lista para que sea una sola lista de
puntos
        return puntos_mas_cercanos(lista)#Retorno el resultado de la llama a
la funcion de puntos mas cercanos de la lista que contiene todos los puntos
    ##########################################################################
#######################################
```

Archivo “algoritmos.py”:

```
import math

def distancia_euclidiana(x_1, y_1, x_2, y_2):
    a = (x_2 - x_1)*(x_2 - x_1)
    b = (y_2 - y_1)*(y_2 - y_1)

    c = a + b

    distancia = math.sqrt(c)

    return distancia

##########################################################################
######
def puntos_mas_cercanos(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
##########################################################################
###########
```

Archivo “mainwindow.py”:

```
from PySide2.QtWidgets import
QMainWindow,QFileDialog,QMessageBox,QTableWidgetItem, QGraphicsScene
from ui_mainwindow import Ui_MainWindow
from administradora import Administradora
from particula import Particula
from PySide2.QtCore import Slot
from PySide2.QtGui import QPen,QColor,QTransform


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

        self.administrador = Administradora()

        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.Agregar_final.clicked.connect(self.agregar_final)
        self.ui.Agregar_Inicio.clicked.connect(self.agregar_inicio)
        self.ui.Mostrar.clicked.connect(self.ver)

        self.ui.actionAbrir.triggered.connect(self.action_abrir_archivo)
        self.ui.actionGuardar.triggered.connect(self.action_guardar_archivo)

        self.ui.view_button.clicked.connect(self.mostrar_tabla)
        self.ui.search_button.clicked.connect(self.buscar_tabla)

        self.ui.Dibujar.clicked.connect(self.dibujar)
        self.ui.Limpiar.clicked.connect(self.limpiar)

        self.scene = QGraphicsScene()
        self.ui.graphicsView.setScene(self.scene)

        self.ui.OrdenarDistancia.clicked.connect(self.ordenarDistancia)
```

```
        self.ui.OrdenarID.clicked.connect(self.ordenarID)

        self.ui.OrdenarDistancia2.clicked.connect(self.ordenarDistancia2)
        self.ui.OrdenarID2.clicked.connect(self.ordenarID2)

        self.ui.DPuntos.clicked.connect(self.DibujarPuntos)
        self.ui.FBruta.clicked.connect(self.FuerzaBruta)

    @Slot()
    def DibujarPuntos(self):
        self.scene.clear()
        pen = QPen()
        pen.setWidth(2)
        for particula in self.administrador:
            origenx = int(particula.origen_x)
            origeny = int(particula.origen_y)
            destinox = int(particula.destino_x)
            destinoy = int(particula.destino_y)
            red = int(particula.red)
            green = int(particula.green)
            blue= int(particula.blue)

            color = QColor(red,green,blue)
            pen.setColor(color)

            self.scene.addEllipse(origenx,origeny,3,3,pen)
            self.scene.addEllipse(destinox,destinoy,3,3,pen)

    ##########################################################################
##################
    @Slot()
    def FuerzaBruta(self):
        pen = QPen()
        pen.setWidth(2)
        resultado = self.administrador.PC()

        for punto1,punto2 in resultado:
            x1 = punto1[0]
            y1 = punto1[1]
            x2 = punto2[0]
            y2 = punto2[1]

            self.scene.addLine(x1,y1,x2,y2,pen)
    ##########################################################################
###################
```

```
    @Slot()
    def ordenarDistancia(self):
        self.ui.Print.clear()
        self.administrador.ordenarDistancia()
        self.ui.Print.insertPlainText(str(self.administrador))

    @Slot()
    def ordenarID(self):
        self.ui.Print.clear()
        self.administrador.ordenarID()
        self.ui.Print.insertPlainText(str(self.administrador))

    @Slot()
    def ordenarDistancia2(self):
        self.ui.table.clear()
        self.administrador.ordenarDistancia()
        self.ui.table.setColumnCount(9)
        headers = ["ID","Origen X","Origen Y","Destino X","Destino
Y","Red","Green","Blue","Distancia"]
        self.ui.table.setHorizontalHeaderLabels(headers)

        self.ui.table.setRowCount(len(self.administrador))

        row = 0
        for particula in self.administrador:
            id_widget = QTableWidgetItem(str(particula.id))
            origenx_widget = QTableWidgetItem(str(particula.origen_x))
            origeny_widget = QTableWidgetItem(str(particula.origen_y))
            destinox_widget = QTableWidgetItem(str(particula.destino_x))
            destinoy_widget = QTableWidgetItem(str(particula.destino_y))
            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.table.setItem(row,0,id_widget)
            self.ui.table.setItem(row,1,origenx_widget)
            self.ui.table.setItem(row,2,origeny_widget)
            self.ui.table.setItem(row,3,destinox_widget)
            self.ui.table.setItem(row,4,destinoy_widget)
            self.ui.table.setItem(row,5,red_widget)
            self.ui.table.setItem(row,6,green_widget)
```

```
            self.ui.table.setItem(row,7,blue_widget)
            self.ui.table.setItem(row,8,distancia_widget)

            row += 1

    @Slot()
    def ordenarID2(self):
        self.ui.table.clear()
        self.administrador.ordenarID()
        self.ui.table.setColumnCount(9)
        headers = ["ID","Origen X","Origen Y","Destino X","Destino
Y","Red","Green","Blue","Distancia"]
        self.ui.table.setHorizontalHeaderLabels(headers)

        self.ui.table.setRowCount(len(self.administrador))

        row = 0
        for particula in self.administrador:
            id_widget = QTableWidgetItem(str(particula.id))
            origenx_widget = QTableWidgetItem(str(particula.origen_x))
            origeny_widget = QTableWidgetItem(str(particula.origen_y))
            destinox_widget = QTableWidgetItem(str(particula.destino_x))
            destinoy_widget = QTableWidgetItem(str(particula.destino_y))
            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.table.setItem(row,0,id_widget)
            self.ui.table.setItem(row,1,origenx_widget)
            self.ui.table.setItem(row,2,origeny_widget)
            self.ui.table.setItem(row,3,destinox_widget)
            self.ui.table.setItem(row,4,destinoy_widget)
            self.ui.table.setItem(row,5,red_widget)
            self.ui.table.setItem(row,6,green_widget)
            self.ui.table.setItem(row,7,blue_widget)
            self.ui.table.setItem(row,8,distancia_widget)

            row += 1




    @Slot()
```

```
    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.administrador:
            origenx = int(particula.origen_x)
            origeny = int(particula.origen_y)
            destinox = int(particula.destino_x)
            destinoy = int(particula.destino_y)

            red = int(particula.red)
            green = int(particula.green)
            blue= int(particula.blue)

            color = QColor(red,green,blue)
            pen.setColor(color)

            self.scene.addEllipse(origenx,origeny,3,3,pen)
            self.scene.addEllipse(destinox,destinoy,3,3,pen)
            self.scene.addLine(origenx,origeny,destinox,destinoy,pen)



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

    @Slot()
    def buscar_tabla(self):
        id = self.ui.search_line.text()
        encontrado = False

        for particula in self.administrador:

            if int(id)  ==  particula.id:
                self.ui.table.clear()
```

```
                self.ui.table.setRowCount(1)
                headers = ["ID","Origen X","Origen Y","Destino X","Destino
Y","Red","Green","Blue","Distancia"]
                self.ui.table.setHorizontalHeaderLabels(headers)

                id_widget = QTableWidgetItem(str(particula.id))
                origenx_widget = QTableWidgetItem(str(particula.origen_x))
                origeny_widget = QTableWidgetItem(str(particula.origen_y))
                destinox_widget = QTableWidgetItem(str(particula.destino_x))
                destinoy_widget = QTableWidgetItem(str(particula.destino_y))
                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.table.setItem(0,0,id_widget)
                self.ui.table.setItem(0,1,origenx_widget)
                self.ui.table.setItem(0,2,origeny_widget)
                self.ui.table.setItem(0,3,destinox_widget)
                self.ui.table.setItem(0,4,destinoy_widget)
                self.ui.table.setItem(0,5,red_widget)
                self.ui.table.setItem(0,6,green_widget)
                self.ui.table.setItem(0,7,blue_widget)
                self.ui.table.setItem(0,8,distancia_widget)

                encontrado = True

                return

        if not encontrado:
            QMessageBox.warning(self,'Atencíon',f'La particula con ID "{id}"
no fue encontrado')




    @Slot()
    def mostrar_tabla(self):
        self.ui.table.setColumnCount(9)
        headers = ["ID","Origen X","Origen Y","Destino X","Destino
Y","Red","Green","Blue","Distancia"]
        self.ui.table.setHorizontalHeaderLabels(headers)
```

```
        self.ui.table.setRowCount(len(self.administrador))

        row = 0
        for particula in self.administrador:
            id_widget = QTableWidgetItem(str(particula.id))
            origenx_widget = QTableWidgetItem(str(particula.origen_x))
            origeny_widget = QTableWidgetItem(str(particula.origen_y))
            destinox_widget = QTableWidgetItem(str(particula.destino_x))
            destinoy_widget = QTableWidgetItem(str(particula.destino_y))
            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.table.setItem(row,0,id_widget)
            self.ui.table.setItem(row,2,origeny_widget)
            self.ui.table.setItem(row,3,destinox_widget)
            self.ui.table.setItem(row,4,destinoy_widget)
            self.ui.table.setItem(row,5,red_widget)
            self.ui.table.setItem(row,6,green_widget)
            self.ui.table.setItem(row,7,blue_widget)
            self.ui.table.setItem(row,8,distancia_widget)

            row += 1


    @Slot()
    def action_abrir_archivo(self):
        ubicacion = QFileDialog.getOpenFileName(self,'Abrir
Archivo','.','JSON (*.json)')[0]
        if self.administrador.abrir(ubicacion):
            QMessageBox.information(self,"Exito","Se abrió el archivo de" +
ubicacion)
        else:
            QMessageBox.information(self,"Error","No se pudo abrir el
archivo de " + ubicacion)


    @Slot()
    def action_guardar_archivo(self):
        ubicacion = QFileDialog.getSaveFileName(self,'Guardar
Archivo','.','JSON (*.json)')[0]
        if self.administrador.guardar(ubicacion):
```

```
            QMessageBox.information(self,"Exito","Se creó el archivo con
exito en " + ubicacion)
        else:
            QMessageBox.information(self,"Error","No se pudo crear el
archivo en " + ubicacion)



    @Slot()
    def ver(self):
        self.ui.Print.clear()
        self.ui.Print.insertPlainText(str(self.administrador))



    @Slot()
    def agregar_final(self):
        ID = self.ui.ID_spinBox.value()
        OrigenX = self.ui.OrigenX_spinBox.value()
        OrigenY = self.ui.OrigenY_spinBox.value()
        DestinoX = self.ui.DestinoX_spinBox.value()
        DestinoY = self.ui.DestinoY_spinBox.value()
        Red = self.ui.Red_spinBox.value()
        Green = self.ui.Green_spinBox.value()
        Blue = self.ui.Blue_spinBox.value()


        particula1 =
Particula(ID,OrigenX,OrigenY,DestinoX,DestinoY,Red,Green,Blue)
        self.administrador.agregar_final(particula1)

    @Slot()
    def agregar_inicio(self):
        ID = self.ui.ID_spinBox.value()
        OrigenX = self.ui.OrigenX_spinBox.value()
        OrigenY = self.ui.OrigenY_spinBox.value()
        DestinoX = self.ui.DestinoX_spinBox.value()
        DestinoY = self.ui.DestinoY_spinBox.value()
        Red = self.ui.Red_spinBox.value()
        Green = self.ui.Green_spinBox.value()
        Blue = self.ui.Blue_spinBox.value()

        particula1 =
Particula(ID,OrigenX,OrigenY,DestinoX,DestinoY,Red,Green,Blue)
        self.administrador.agregar_inicio(particula1)
```

Archivo “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,red = 0, green = 0, blue = 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.__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 en X :' +
str(self.__origen_x) + '\n' +
               'Origen en Y :' + str(self.__origen_y) + '\n' + 'Destino en X
:' + str(self.__destino_x) + '\n' +
               'Destino en Y: ' + str(self.__destino_y) + '\n' + 'Distancia
: ' + str(self.__distancia) + '\n' +
               'Red :' + str(self.__red) + '\n' 'Green :' +
str(self.__green) + '\n' 'Blue :' + str(self.__blue) + '\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 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_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,
            "red": self.__red,
            "green": self.__green,
            "blue": self.__blue
        }
```

Archivo “prueba.py”:

```
from PySide2.QtWidgets import QApplication
from mainwindow import MainWindow
import sys

app = QApplication()

window = MainWindow()

window.show()

sys.exit(app.exec_())
```