

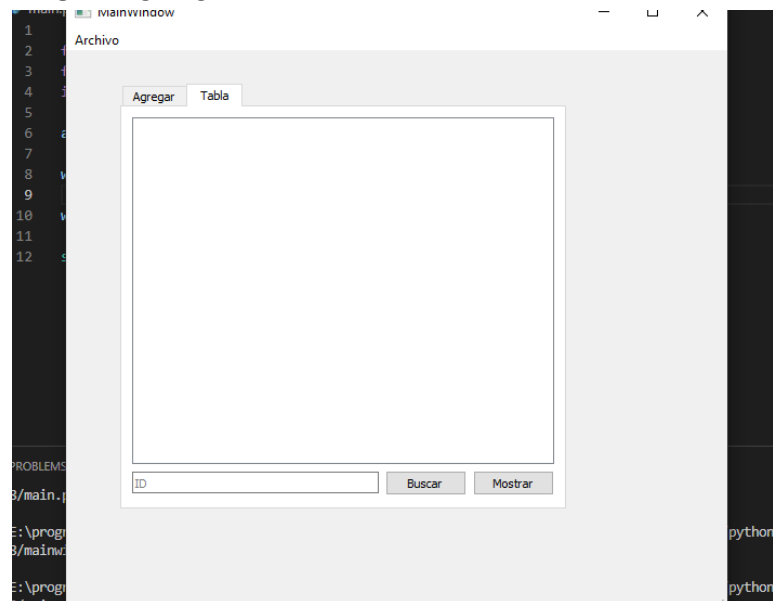
Actividad 08 (QWidget)

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 la captura de pantalla de las partículas con el método mostrar() previo a generar el respaldo.
- [] Se muestran capturas de pantallas de los pasos que se realizan en la interfaz para generar el respaldo.
- [] Se muestra el contenido del archivo *.json*.
- [] Se muestran capturas de pantallas de los pasos que se realizan en la interfaz para abrir el archivo de respaldo *.json*.
- [] Se muestra la captura de pantalla de las partículas con el método mostrar() después de abrir el respaldo.

DESARROLLO:



Agregar

Tabla

	id	origen_x	origen_y	destino_x
1	7	4	4	4
2	3	0	0	0
3	7	0	0	0

Agregar

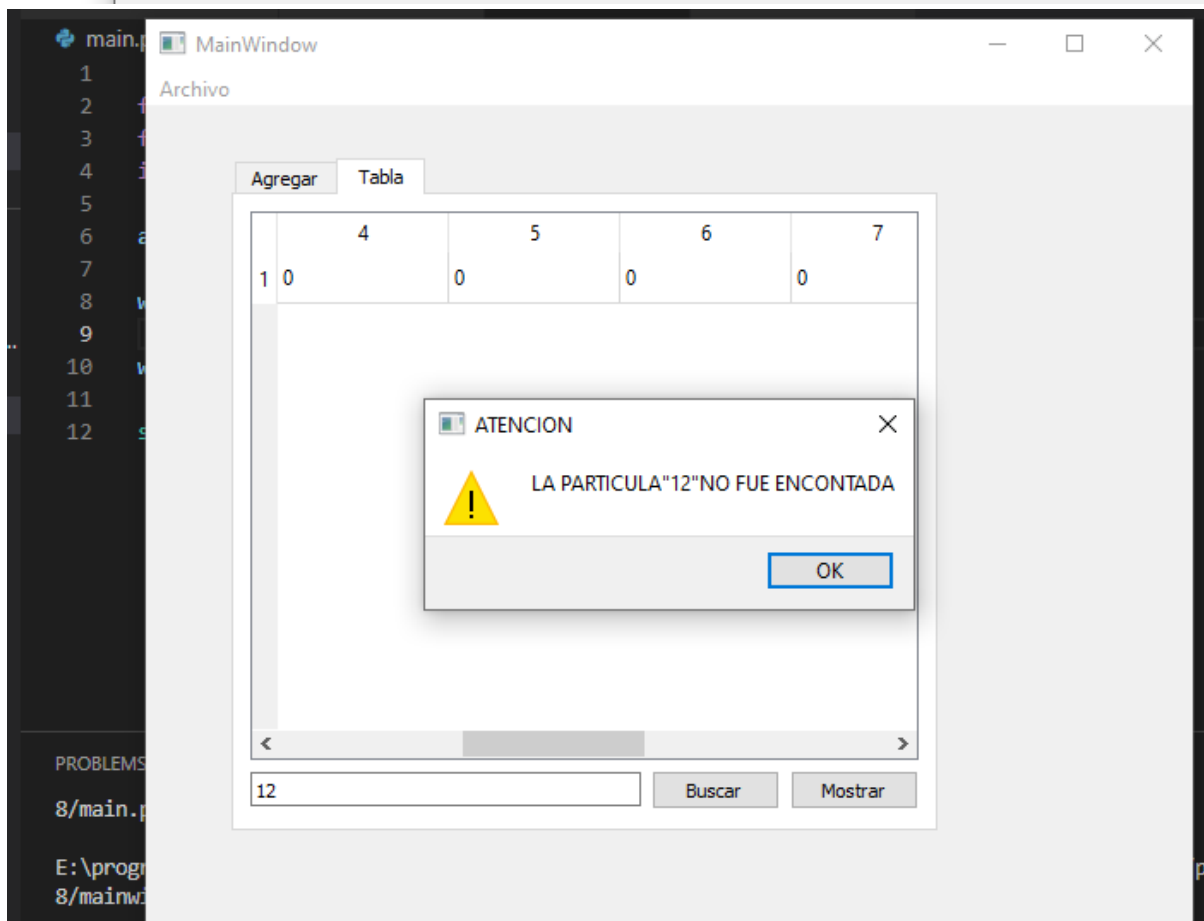
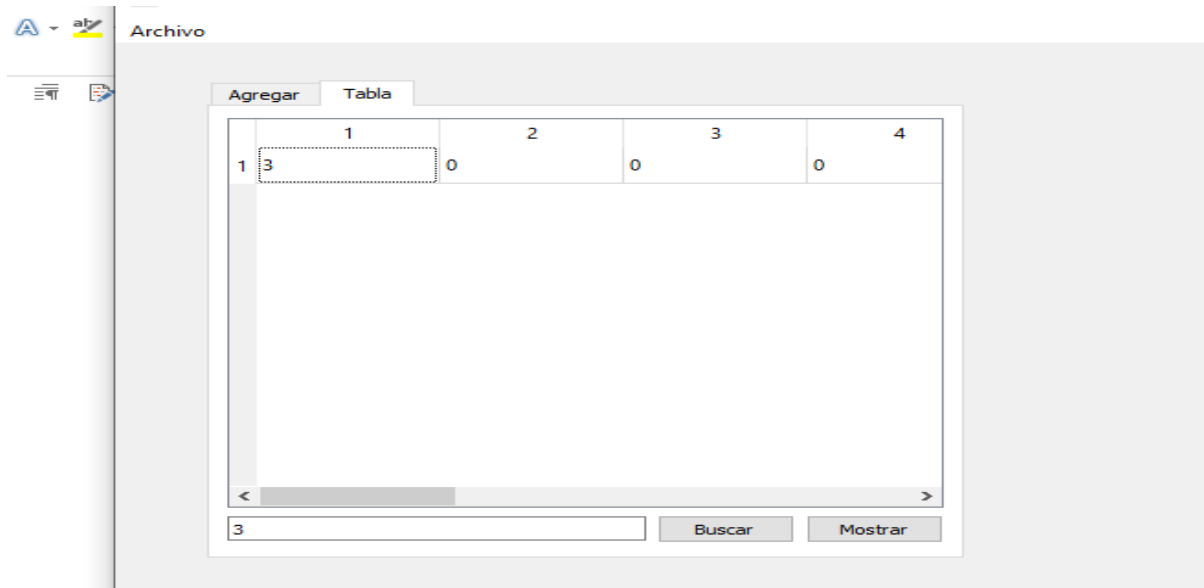
Tabla

	origen_y	destino_x	destino_y	velocidad
1	4	4	0	0
2	0	0	0	0
3	0	0	0	0

Agregar

Tabla

	red	green	blue	distancia
1	0	0	0	4.0
2	0	0	0	0.0
3	0	0	0	0.0



Código:

```
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
from PySide2.QtCore import Slot
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)

    @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", "distancia"]
    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 ABRIÓ CON ÉXITO EL ARCHIVO" + ubicacion
        )
    else:
        QMessageBox.critical(
            self,
            "ERROR",
            "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()

    particula =
Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,rojo,verde,azu
1)
        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
1)
        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 __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)
                self.__Lista = [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 < 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,
        }

    @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 crear una tabla y colocar en ella los datos de mi clase partícula así también como a generar un buscador dentro de la tabla

Referencias:

<https://youtu.be/1yEpAHaiMxs>