Actividad 08 (QTableWidget)

Jose Eduardo Silva Canizales

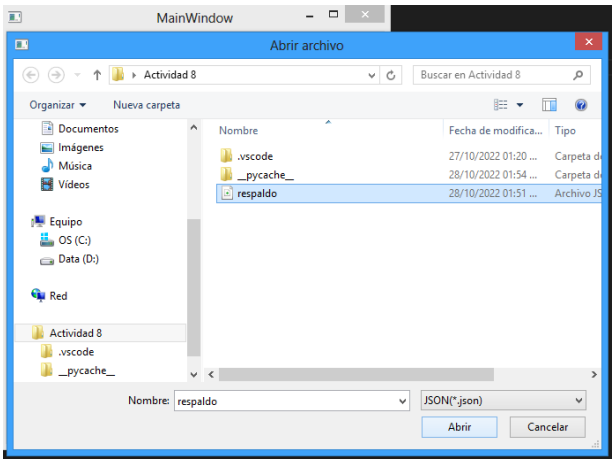
Seminario de solución de problemas de algoritmia

Lineamientos de evaluación

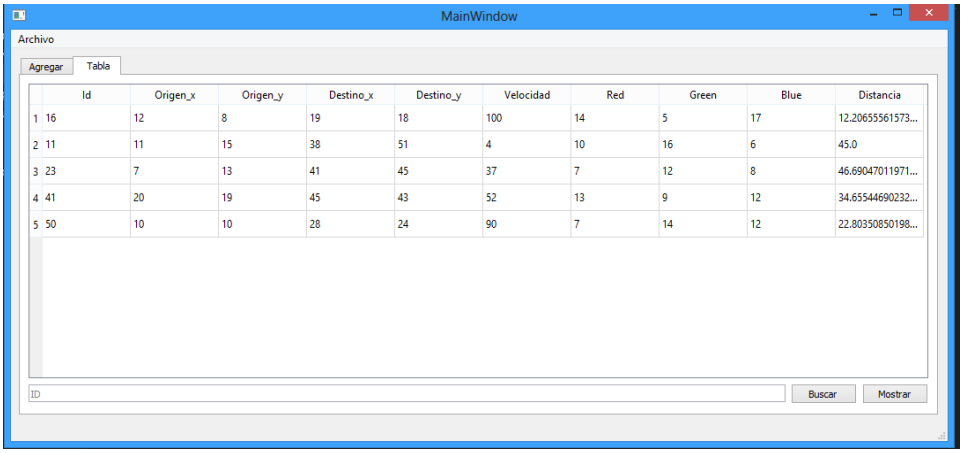
- El reporte está en formato Google Docs o PDF.
- El reporte sigue las pautas del Formato de Actividades.
- El reporte tiene desarrollada todas las pautas del Formato de Actividades.
- Se muestra captura de pantalla de lo que se pide en el punto 2. sub punto a.
- Se muestra captura de pantalla de lo que se pide en el punto 2. sub punto b.
- Se muestra captura de pantalla de lo que se pide en el punto 2. sub punto c.
- Se muestra captura de pantalla de lo que se pide en el punto 2. sub punto d.

Desarrollo

Captura de pantalla de recuperación de archivo.

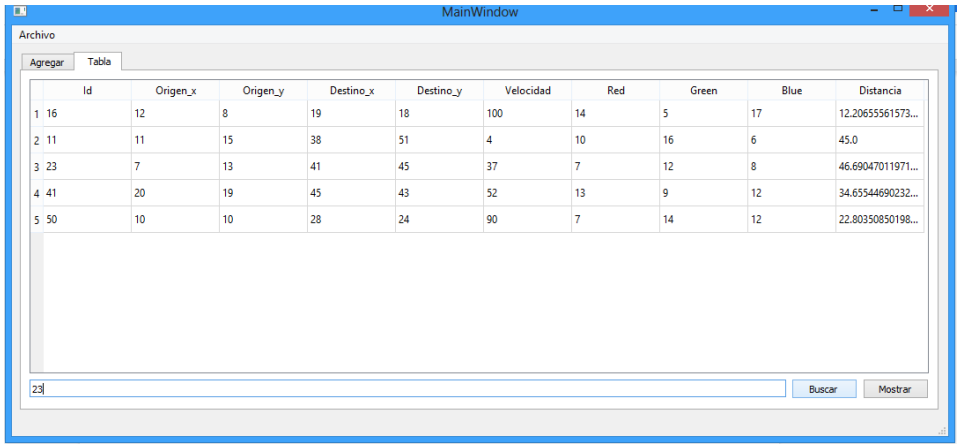


Captura de pantalla de las partículas en el QTableWidget.

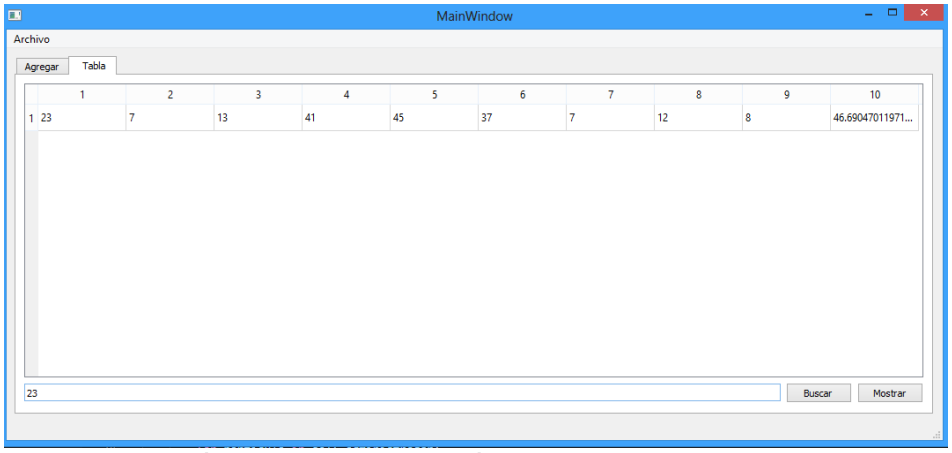


Captura de pantalla de la búsqueda de una partícula con un id existente.

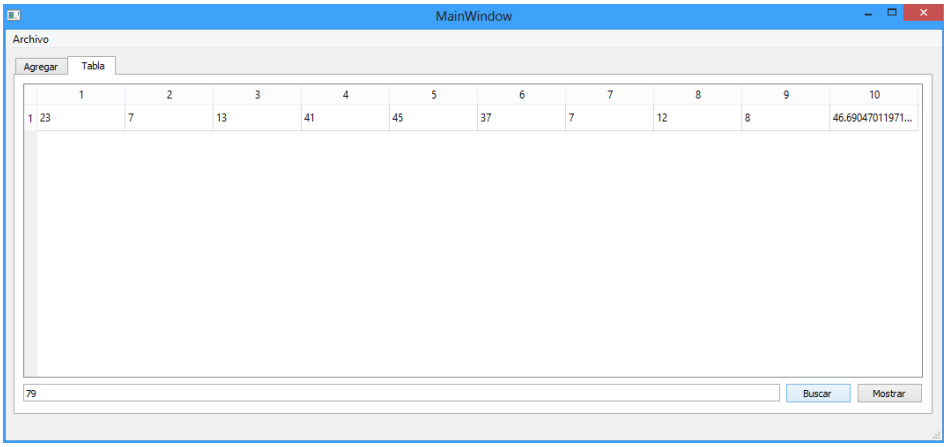
Ingresando el id.



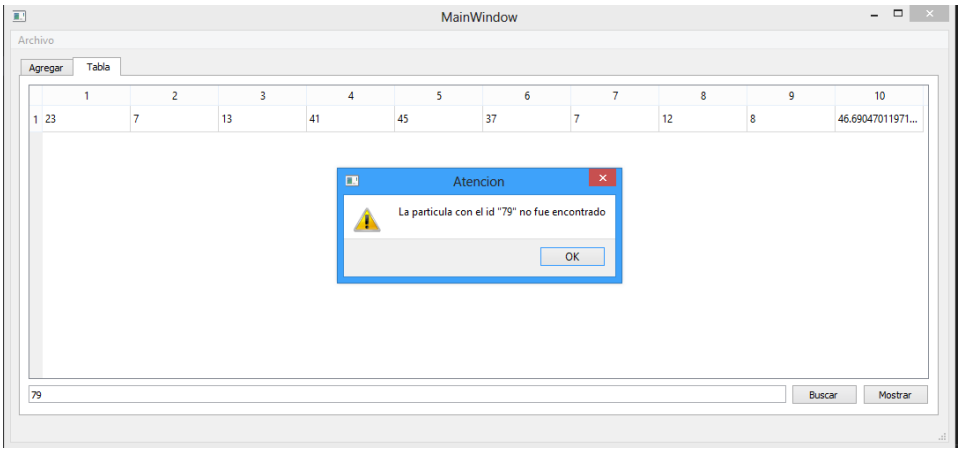
Búsqueda realizada.



Captura de pantalla de la búsqueda de una partícula con un id no existente. Ingresando el id.



Búsqueda realizada.



Conclusiones

Durante esta actividad aprendí a hacer que la información que se ingrese o se recupero sea visualizada en el QTableWidget, dándole un mayor orden a los datos, por otra parte aprendí a hacer búsquedas a través der un id, mostrando únicamente la fila que contiene los datos.

También en un inicio tuve problemas en que se hiciera la búsqueda correctamente, pero mediante el video de referencia pude corregir el problema lo que llevo a tener la búsqueda de manera correcta.

Referencias

Primera referencia

Url: https://www.youtube.com/watch?v=1yEpAHaiMxs

Título: PySide2 - QTableWidget (Qt for Python)(V)

Autor: MICHEL DAVALOS BOITES

Código

main.py

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

# Aplicacion de Qt
app = QApplication()

# Se crea window
window = MainWindow()

#se hace visible window
window.show()
# Qt loop
sys.exit(app.exec_())
```

mainwindow.py

```
from wsgiref import headers
from PySide2.QtWidgets import
QMainWindow,QFileDialog,QMessageBox,QTableWidgetItem
from PySide2.QtCore import Slot
from ui_mainwindow import Ui_MainWindow
from particula import Particula
from administrador import Adminisrador

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

        self.administrador=Adminisrador()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.agregar_final_pushButton.clicked.connect(
            self.click_agregar_final)
        self.ui.agregar_inicio_pushButton.clicked.connect(
            self.click_agregar_inicio)
        self.ui.mostar_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):
        #value
        id = self.ui.buscar_lineEdit.text()
        encontrado=False
```

```
        for particula in self.administrador:
            if id== particula.id:
                self.ui.tabla.clear()
                self.ui.tabla.setRowCount(1)

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

                encontrado=True

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




    @Slot()
    def mostrar_tabla(self):
        self.ui.tabla.setColumnCount(10)
        headers =
["Id","Origen_x","Origen_y","Destino_x","Destino_y","Velocidad","Red","Green
","Blue","Distancia"]
        self.ui.tabla.setHorizontalHeaderLabels(headers)

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

```
    row=0
    for particula in self.administrador:

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

        row+=1
```

```
@Slot()
def action_abrir_archivo(self):
    ubicacion=QFileDialog.getOpenFileName(
        self,
        'Abrir archivo',
        '.',
        'JSON(*.json)'
    )[0]
    print(ubicacion)
    if self.administrador.abrir(ubicacion):
        QMessageBox.information(
            self,
            "Exito",
            "se cargo el archivo" + ubicacion
        )
    else:
        QMessageBox.critical(
            self,
            "Error",
```

```
                "Error al abrir el archivo" + ubicacion
            )


    @Slot()
    def action_guardar_archivo(self):
        #print('guardar_archivo')
        ubicacion=QFileDialog.getSaveFileName(
            self,
            'Guardar',
            '.',
            'JSON(*.json)'
        )[0]
        print(ubicacion)
        if self.administrador.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.clear()
        self.ui.salida.insertPlainText(str(self.administrador))

    @Slot()
    def click_agregar_final(self):
        id = self.ui.id_lineEdit.text()
        origen_x = self.ui.origen_x_spinBox.value()
        origen_y = self.ui.origen_y_spinBox.value()
        destino_x = self.ui.destino_x_spinBox.value()
        destino_y = self.ui.destino_y_spinBox.value()
        velocidad = self.ui.velocidad_spinBox.value()
        red = self.ui.red_spinBox.value()
        green = self.ui.green_spinBox.value()
        blue = self.ui.blue_spinBox.value()
        distancia = self.ui.distancia_spinBox.value()

        particula=Particula(id,origen_x,origen_y,destino_x,destino_y,velocid
ad,red,green,blue,distancia)
        self.administrador.agregar_final(particula)

    @Slot()
```

```
    def click_agregar_inicio(self):
        id = self.ui.id_lineEdit.text()
        origen_x = self.ui.origen_x_spinBox.value()
        origen_y = self.ui.origen_y_spinBox.value()
        destino_x = self.ui.destino_x_spinBox.value()
        destino_y = self.ui.destino_y_spinBox.value()
        velocidad = self.ui.velocidad_spinBox.value()
        red = self.ui.red_spinBox.value()
        green = self.ui.green_spinBox.value()
        blue = self.ui.blue_spinBox.value()
        distancia = self.ui.distancia_spinBox.value()

        particula=Particula(id,origen_x,origen_y,destino_x,destino_y,velocid
ad,red,green,blue,distancia)
        self.administrador.agregar_inicio(particula)
```

administrador.py

```
from algoritmos import distancia_euclidiana
from particula import Particula
import json

class Adminisrador:
    def __init__(self):
        self.__particulas = []

    def agregar_final(self, particula: Particula):
        self.__particulas.append(particula)

    def agregar_inicio(self, particula: Particula):
        self.__particulas.insert(0, particula)

    def mostrar(self):
        for particula in self.__particulas:
            print(particula)

    def __str__(self):
        return "".join(
            str(particula)+'\n' for particula in self.__particulas
        )
    def guardar(self,ubicacion):
        try:
            with open(ubicacion, 'w') as archivo:
                lista = [particula.to_dict() for particula in
self.__particulas]
                print(lista)
                json.dump(lista, archivo, indent=5)
            return 1
        except :
            return 0

    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 __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
```

particula.py

```
from algoritmos import distancia_euclidiana

class Particula:
    def __init__(self,
                 id="", origen_x=0, origen_y=0, destino_x=0,
destino_y=0,velocidad=0,red=0,green=0,blue=0,distancia=0.0):

        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'
        )

@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 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,
        "velocidad": self.__velocidad,
```

```
        "red": self.__red,
        "green": self.__green,
        "blue": self.__blue
    }
```