Actividad 08 – (QTableWidget)

Rubio Valenzuela Miguel Angel - 216567795

Seminario de Algoritmia – D02.

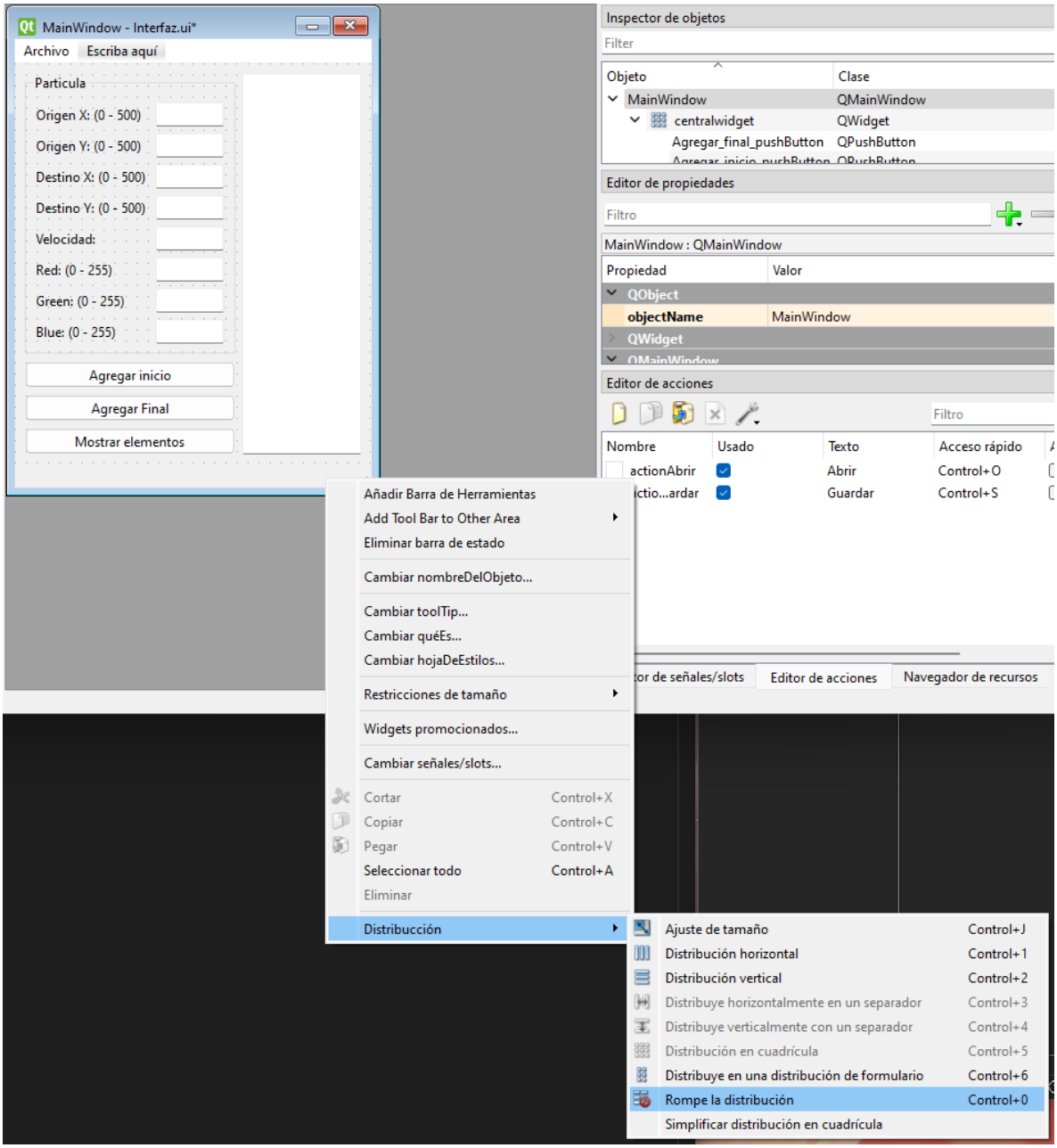
Lineamientos de evaluación

- [ ] 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.

Para poder realizar la actividad necesitamos el componente QTableWidget, donde mostraremos todos los datos que ya tenemos en una tabla.

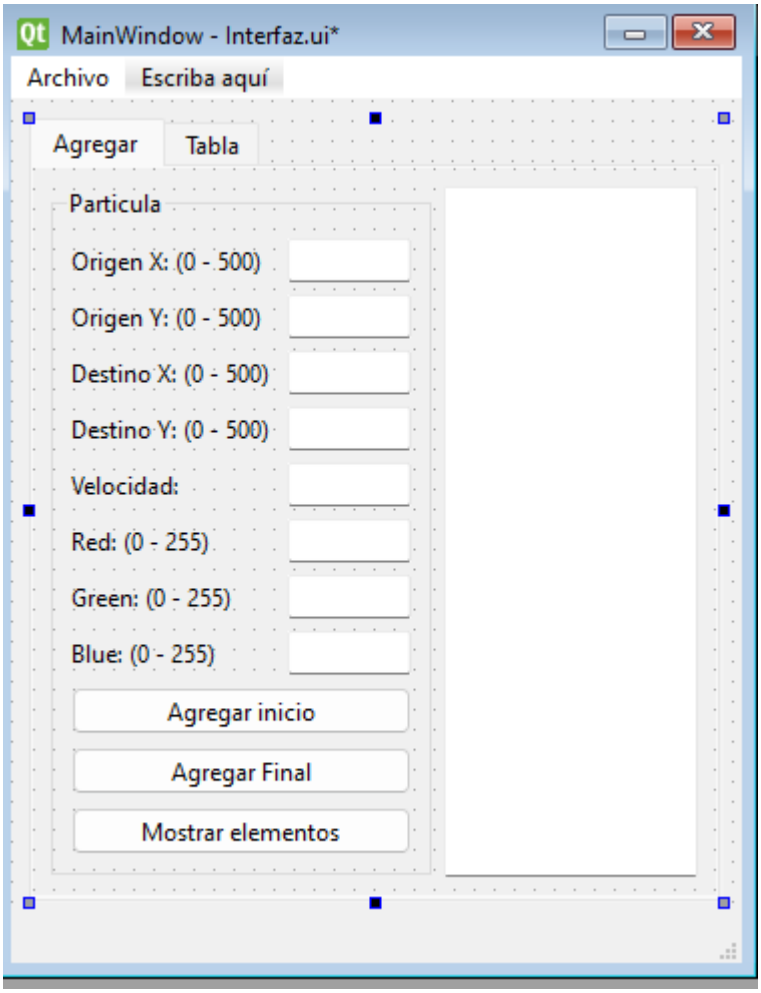
Lo primero que tenemos que realizar es el cambio de distribución de nuestra interfaz de usuario, para esto tenemos que entrar en el Qt Designer y romper la distribución que estaba manejando y ahí vamos a tener que realizar el ajuste.



Después de realizar el rompimiento de la distribución lo que tenemos que hacer es ingresar las pestañas.

Para esto tenemos que añadir un Tab Widget donde vamos a tener dos pestañas la pestaña de agregar y la otra pestaña donde tenemos la tabla, la primera donde tenemos la pestaña de añadir que quedaría de la siguiente forma, luego de realizar

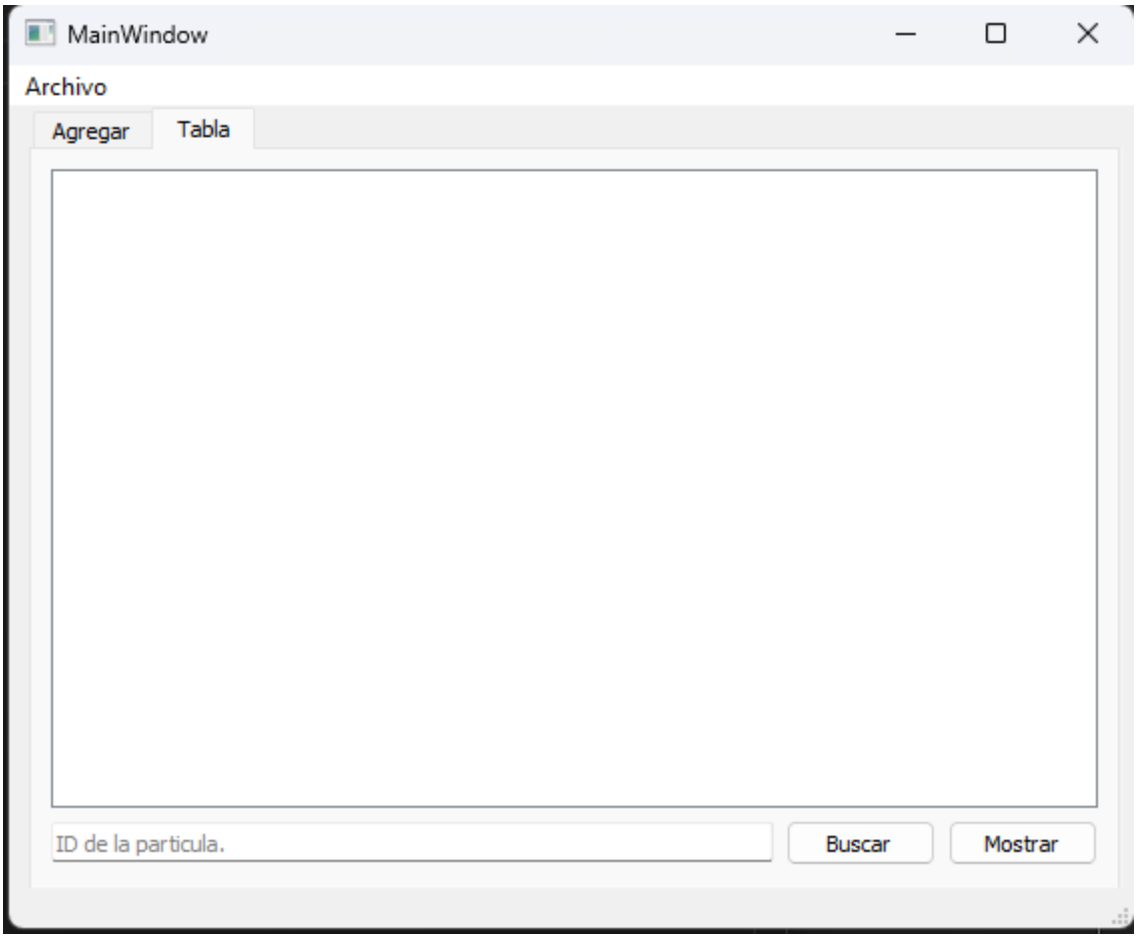
Para cambiar el nombre a las pestañas nosotros tenemos que cambiarle el nombre desde el apartado CurrentTabText, y ahí nosotros vamos a poder ponerle el nombre que nosotros queramos.



Esta es la manera en la que queda la pestaña de agregar.

Ahora tenemos que ingresar la información en la otra pestaña.

Donde nosotros vamos a ingresar una tabla donde se mostraran los datos que nosotros ingresamos, una apartado donde vamos a realizar una búsqueda.



Esta es la manera en la que se ve la interfaz de la tabla.

Lo que tenemos que hacer ahora es darle funcionalidad a este aparatado.

Lo primero que tenemos que hacer es conectar los valores y lo siguiente a hacer es darle esa funcionalidad a la función mostrar.

El código que manda una frase en consola al presionar los botones es el siguiente:

```
    self.ui.mostrar_tabla_pushButton.clicked.connect(self.mostrar_tabla)
    self.ui.buscar_pushButton.clicked.connect(self.buscar_elemento)

@Slot()
def mostrar_tabla(self):
    print("Mostrar tabla")

@Slot()
def buscar_elemento(self):
    print("Buscando")
```

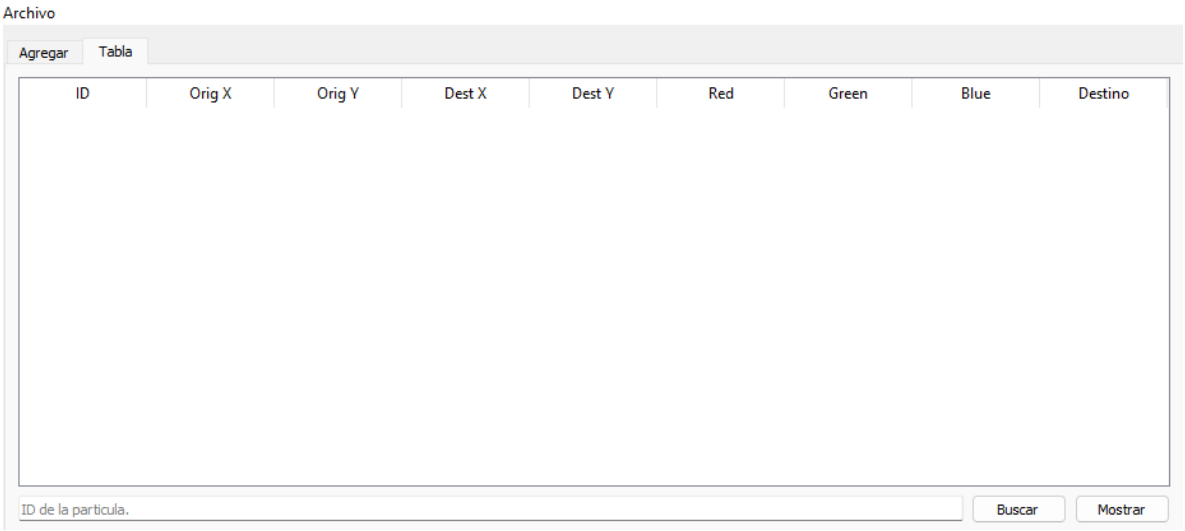
Con esto, podemos presionar cualquiera de los dos botones en nuestra interfaz y podremos mostrar una frase correspondiente en nuestra consola, asi como se muestra a continuación:

```
ensions\ms-python.python-2022.16.1\pythonFiles\l
0549' '--' 'c:\Users\cober\Desktop\Sem de Algori
Buscando
Mostrar tabla
PS C:\Users\cober\Desktop\Sem de Algoritmia\A8>
```

Ahí podemos ver que presionar los botones ya tiene una funcionalidad dada, pero esta no es la funcionalidad que nosotros estamos buscando a la hora de presionar estos botones, así que lo que tenemos que hacer es darle esta funcionalidad.

Para la función mostrar_tabla tenemos que agregar el numero de columnas que debe de tener nuestra tabla usando el método setColumnCount y además utilizando headers para las cabeceras de la tabla, y añadiéndolas con el método setHorizontalHeaderLabels.

Esta es la forma en la que quedo nuestra tabla:



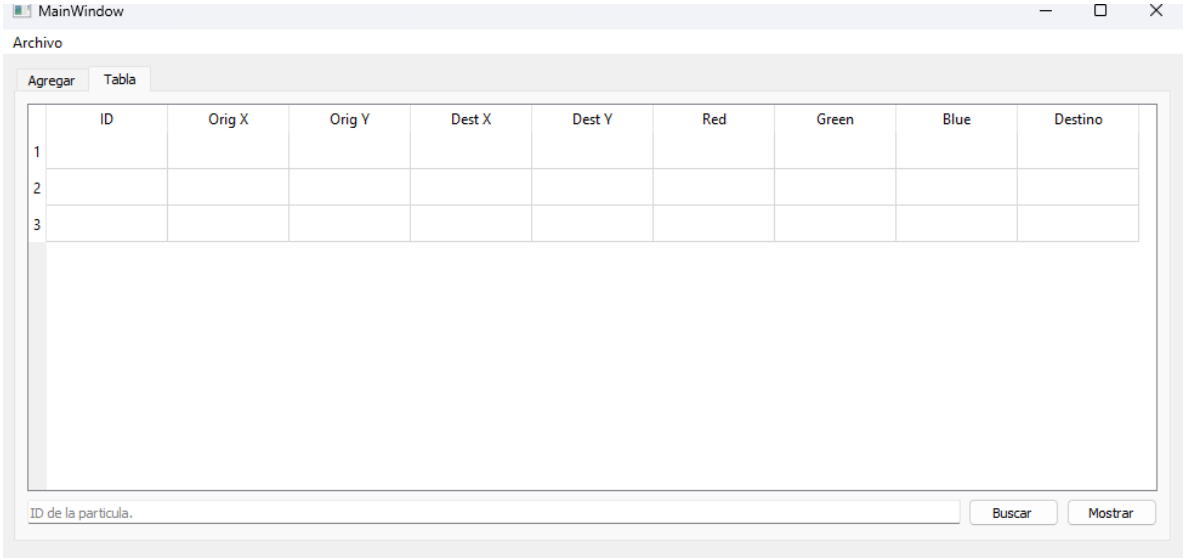
Lo que tenemos que realizar, para poder generar las filas tenemos que usar una función llamada setRowCount, pero para esto tenemos que sacar la cantidad de elementos que tenemos en nuestra lista de partículas, y para esto tenemos que implementar la función len() en la lista de partículas.

Se va a mostrar el numero de libros que va a tener nuestra lista de particulas.

```python
def __len__(self):
    return(len(self.__particulas))
```

Utilizando este código tenemos el numero de elementos que debe de tener respecto a cada uno de los registros ingresados en nuestro programa.

Y esta es la forma en la que queda cuando tenemos 3 registros ingresados.



Podemos ver que tenemos 3 filas y que estas dependen del numero de registros que tenemos ingresados en nuestra lista.

Ahora lo que sigue es obtener cada libro de la lista e ingresarla en la tabla.

Para poder hacer esto tenemos que implementar dos funciones que lo que van a hacer es que nuestra lista sea iterable, y esta es la forma en la que queda:

```
def __iter__(self):
    self.cont = 0
    return self.cont

def __next__(self):
    if self.cont < len(self.__particulas):
        part = self.__particulas[self.cont]
        self.cont += 1
        return part
    else:
        raise StopIteration
```

Iter lo que hace es crear un contador y devolverlo, y lo que hace next es validar si el contador se encuentra dentro de los valores de la lista, y de ser así vamos a devolver todo el objeto que se encuentre en la posición contador.

Ahora gracias a esto, podemos mostrar toda la información que nosotros tenemos en la lista de partículas, pero podemos ver esto en la consola.

Para que la información sea reflejada en la tabla, tenemos que usar una librería que nos va a permitir esto, la cual se llama QTableWidgetItem.

Pero para que nosotros podamos extraer la información y poder ingresarla y mostrarla en la tabla tenemos que realizar los getters y obtener la información que nosotros estamos requiriendo

Estas funciones deben hacerse de forma independiente para cada uno de las variables que nosotros contengamos en el objeto.

```python
@property
def id(self):
    return self.__id

@property
def origenX(self):
    return self.__origen_x

@property
def origenY(self):
    return self.__origen_y

@property
def destinoX(self):
    return self.__destino_x

@property
def destinoY(self):
    return self.__destino_y

@property
def velocidad(self):
    return self.__velocidad

@property
def red(self):
    return self.__red
```

```python
@property
def green(self):
    return self.__green

@property
def blue(self):
    return self.__blue

@property
def distancia(self):
    ox = float(self.__origen_x)
    oy = float(self.__origen_y)
    dx = float(self.__destino_x)
    dy = float(self.__destino_y)
    resultado = sqrt(pow((ox - dx), 2) + pow((oy - dy), 2))
    return str(resultado)
```

En la funcion de distancia lo que hago es el calculo para que me pueda dar el valor de la distancia que nosotros esperamos tener.

Esta es la forma en la que el código debe de quedar para obtener los valores de cada uno de forma independiente

Ahora para que la información se vea en la tabla tenemos que usar un método de nuestra tabla llamado, setItem, donde ahí los valores que necesita son el numero de columna, la fila donde ingresarlo y además el valor que va a tener

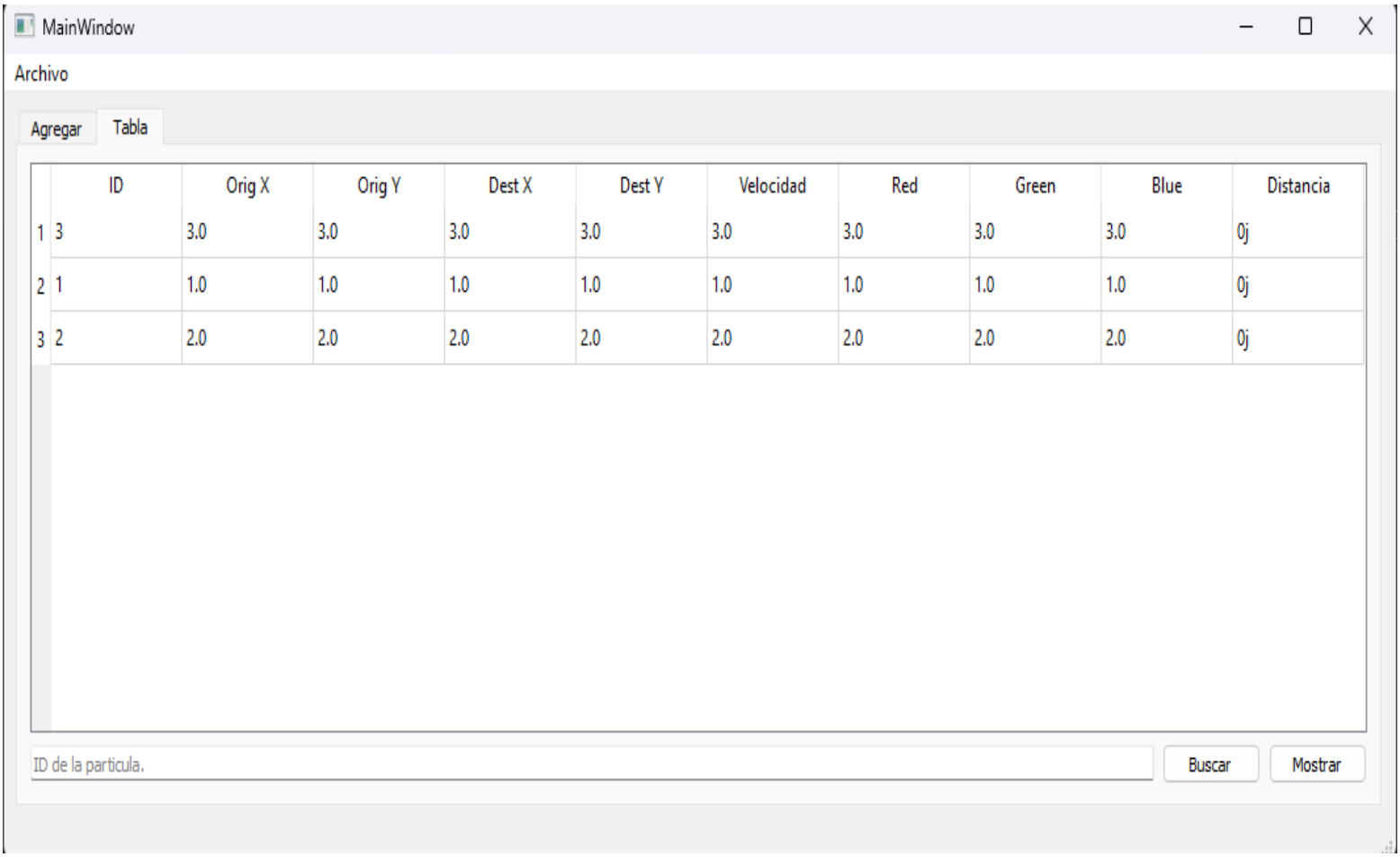
La manera en la que queda la funcion de mostrar tabla es la siguiente:

```python
@Slot()
def mostrar_tabla(self):
    self.ui.tabla.setColumnCount(10)
    headers = ["ID","Orig X", "Orig Y", "Dest X", "Dest Y", "Velocidad" ,"Red",  "Green", "Blue", "Distancia"]
    self.ui.tabla.setHorizontalHeaderLabels(headers)

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

```
row = 0
for particula in self.lista:
    id_widget = QTableWidgetItem(particula.id)
    origenX_widget = QTableWidgetItem(particula.origenX)
    origenY_widget = QTableWidgetItem(particula.origenY)
    destinoX_widget = QTableWidgetItem(particula.destinoX)
    destinoY_widget = QTableWidgetItem(particula.destinoY)
    velocidad_widget = QTableWidgetItem(particula.velocidad)
    red_widget = QTableWidgetItem(particula.red)
    green_widget = QTableWidgetItem(particula.green)
    blue_widget = QTableWidgetItem(particula.blue)
    distancia_widget = QTableWidgetItem(particula.distancia)

    self.ui.tabla.setItem(row, 0, id_widget)
    self.ui.tabla.setItem(row, 1, origenX_widget)
    self.ui.tabla.setItem(row, 2, origenY_widget)
    self.ui.tabla.setItem(row, 3, destinoX_widget)
    self.ui.tabla.setItem(row, 4, destinoY_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
```

Y de esta manera nosotros le estamos ingresando los valores de nuestra lista de partículas a nuestra tabla que mostraremos en la pantalla.

Ahora veremos como es el resultado que nos da al momento de realizar las siguientes acciones, primero abrir el archivo que contiene toda la información de las particulas, y luego mostrarla en la tabla



En la tabla ahora si podemos ver todos los valores que se encuentran en la lista de partículas.

Lo que tenemos que hacer ahora es que el apartado de búsqueda funcione.

Para el apartado de búsqueda lo que tenemos que realizar es lo siguiente, utilizar el mismo código para mostrar, pero agregando una condicional la cual nos va a comparar si el número que nosotros le ingresamos es el mismo que el que se encuentre dentro de los registros.

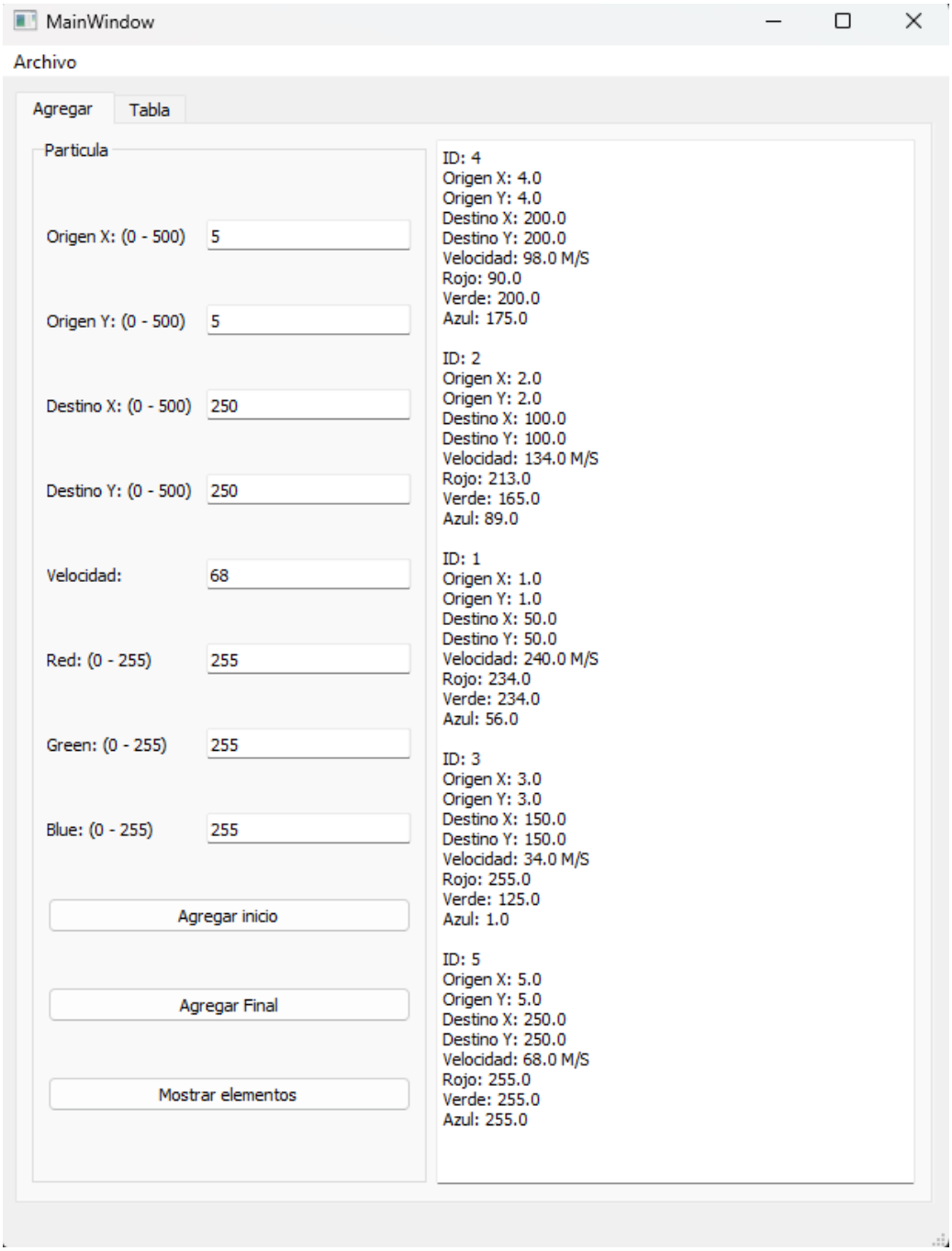
Aquí se muestra la función completa

```python
@Slot()
def buscar_elemento(self):
    identificador = self.ui.buscar_lineEdit.text()

    find = False
    for particula in self.lista:
        if identificador == particula.id:
            self.ui.tabla.clear()
            self.ui.tabla.setColumnCount(10)
            headers = ["ID","Orig X", "Orig Y", "Dest X", "Dest Y",
            "Velocidad" ,"Red",  "Green", "Blue", "Distancia"]
            self.ui.tabla.setHorizontalHeaderLabels(headers)
            self.ui.tabla.setRowCount(1)

            id_widget = QTableWidgetItem(particula.id)
            origenX_widget = QTableWidgetItem(particula.origenX)
            origenY_widget = QTableWidgetItem(particula.origenY)
            destinoX_widget = QTableWidgetItem(particula.destinoX)
            destinoY_widget = QTableWidgetItem(particula.destinoY)
            velocidad_widget = QTableWidgetItem(particula.velocidad)
            red_widget = QTableWidgetItem(particula.red)
            green_widget = QTableWidgetItem(particula.green)
            blue_widget = QTableWidgetItem(particula.blue)
            distancia_widget = QTableWidgetItem(particula.distancia)

            self.ui.tabla.setItem(0, 0, id_widget)
            self.ui.tabla.setItem(0, 1, origenX_widget)
            self.ui.tabla.setItem(0, 2, origenY_widget)
            self.ui.tabla.setItem(0, 3, destinoX_widget)
            self.ui.tabla.setItem(0, 4, destinoY_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)

            find = True
            return

    if not find:
        QMessageBox.warning(
            self,
            "Atencion",
            "La particula" + identificador + "no encontrada"
        )
```

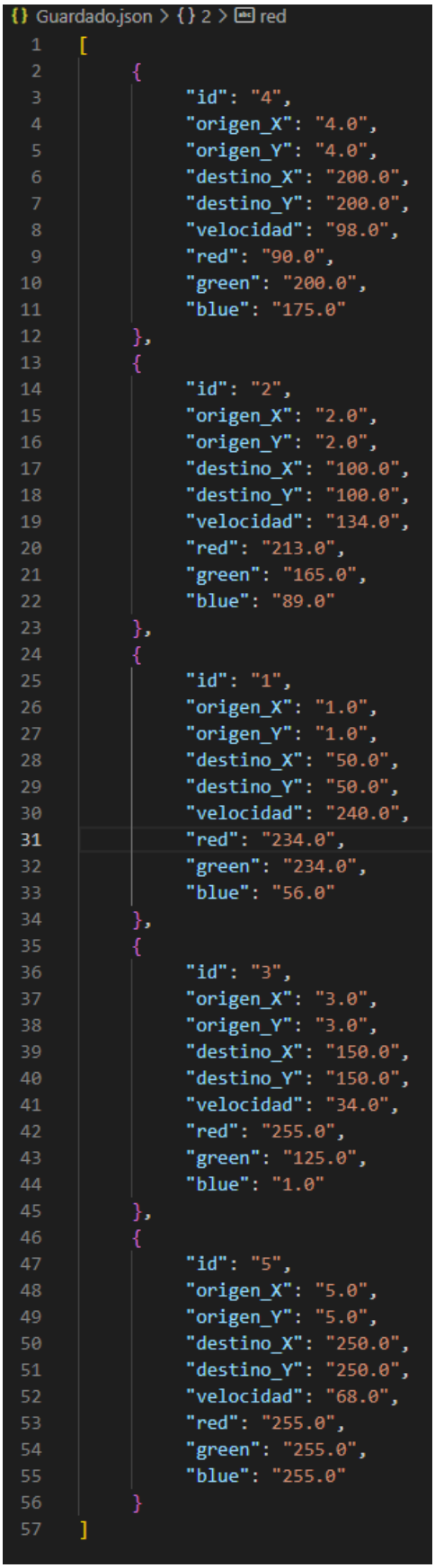
Y con esto, terminamos con el desarrollo del programa

Lo que tenemos que hacer ahora es poner a prueba este código de la siguiente manera:

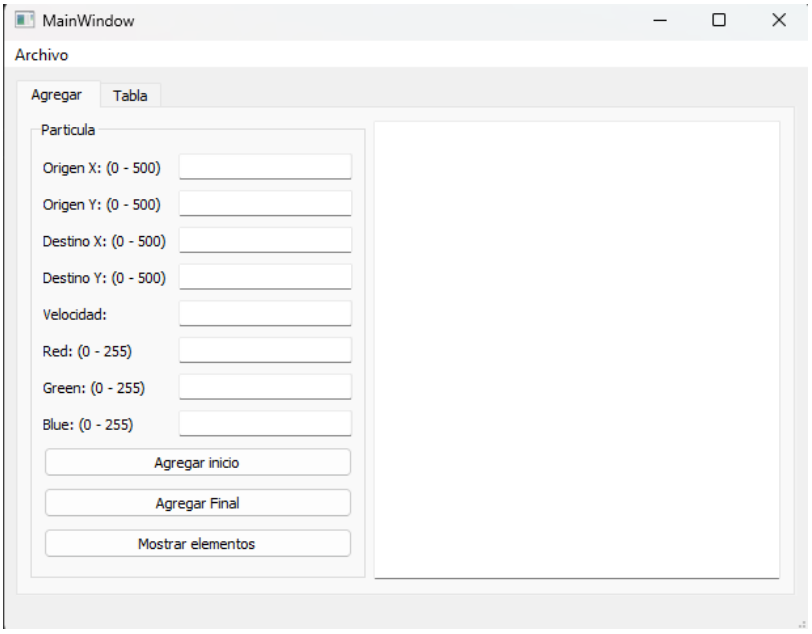
Agregar 5 partículas:



Guardamos esta lista en un archivo llamado, guardado.json

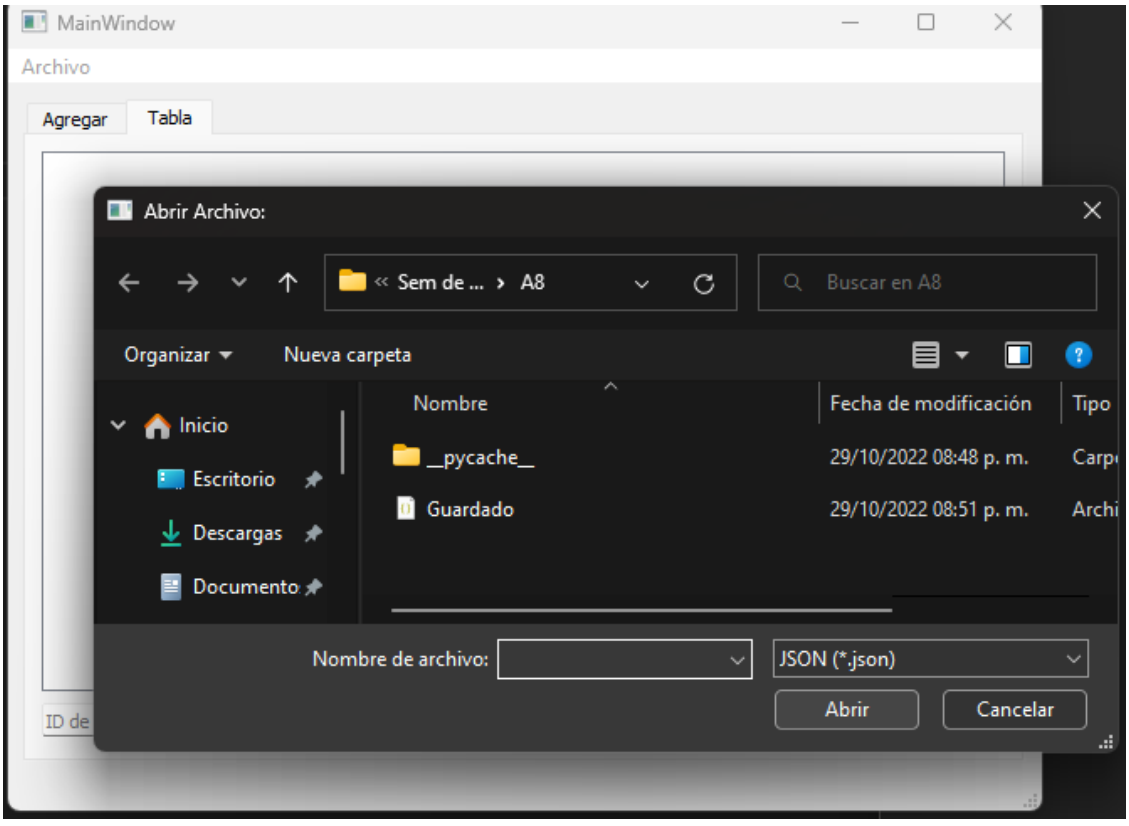

{} Guardado.json > {} 2 > red
[
{
"id": "4",
"origen_X": "4.0",
"origen_Y": "4.0",
"destino_X": "200.0",
"destino_Y": "200.0",
"velocidad": "98.0",
"red": "90.0",
"green": "200.0",
"blue": "175.0"
},
{
"id": "2",
"origen_X": "2.0",
"origen_Y": "2.0",
"destino_X": "100.0",
"destino_Y": "100.0",
"velocidad": "134.0",
"red": "213.0",
"green": "165.0",
"blue": "89.0"
},
{
"id": "1",
"origen_X": "1.0",
"origen_Y": "1.0",
"destino_X": "50.0",
"destino_Y": "50.0",
"velocidad": "240.0",
"red": "234.0",
"green": "234.0",
"blue": "56.0"
},
{
"id": "3",
"origen_X": "3.0",
"origen_Y": "3.0",
"destino_X": "150.0",
"destino_Y": "150.0",
"velocidad": "34.0",
"red": "255.0",
"green": "125.0",
"blue": "1.0"
},
{
"id": "5",
"origen_X": "5.0",
"origen_Y": "5.0",
"destino_X": "250.0",
"destino_Y": "250.0",
"velocidad": "68.0",
"red": "255.0",
"green": "255.0",
"blue": "255.0"
}
]

Podemos observar el archivo que tiene los registros ingresados anteriormente, luego de esto, cerramos el programa y lo volvemos a abrir.

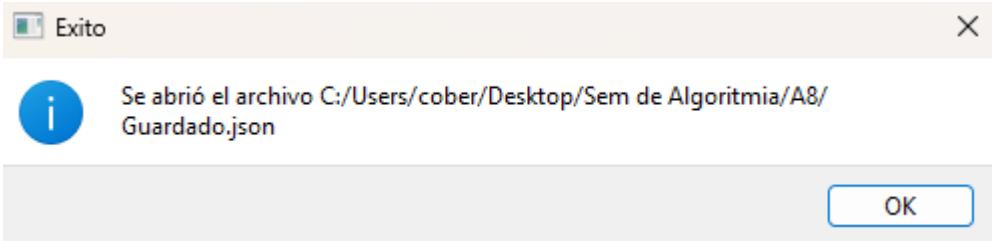


Podemos ver que los registros se encuentran borrados, esto porque volví a abrir el programa, pero no he cargado los datos.

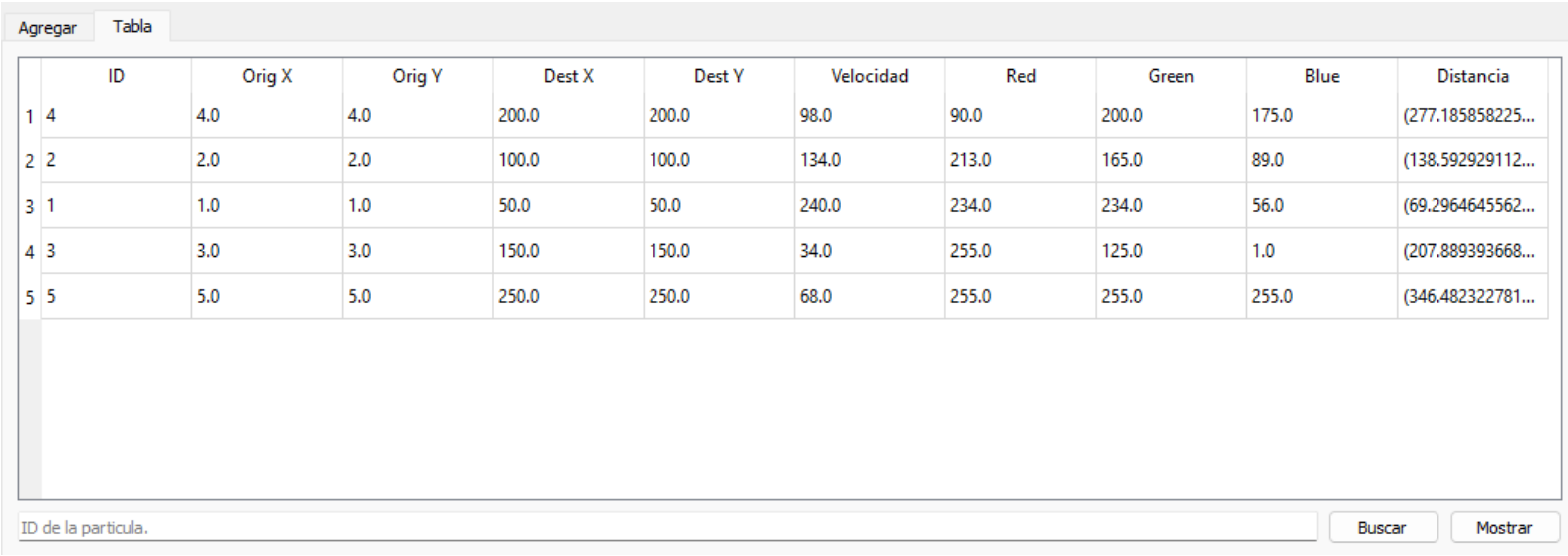
En el apartado tabla, realizamos la apertura del archivo guardado.json



Una vez abierto el archivo, podemos mostrar los datos de la tabla con la información almacenada ahí.



Esta es la tabla luego de dar click en mostrar la información.



	ID	Orig X	Orig Y	Dest X	Dest Y	Velocidad	Red	Green	Blue	Distancia
1	4	4.0	4.0	200.0	200.0	98.0	90.0	200.0	175.0	(277.185858225...
2	2	2.0	2.0	100.0	100.0	134.0	213.0	165.0	89.0	(138.592929112...
3	1	1.0	1.0	50.0	50.0	240.0	234.0	234.0	56.0	(69.2964645562...
4	3	3.0	3.0	150.0	150.0	34.0	255.0	125.0	1.0	(207.889393668...
5	5	5.0	5.0	250.0	250.0	68.0	255.0	255.0	255.0	(346.482322781...

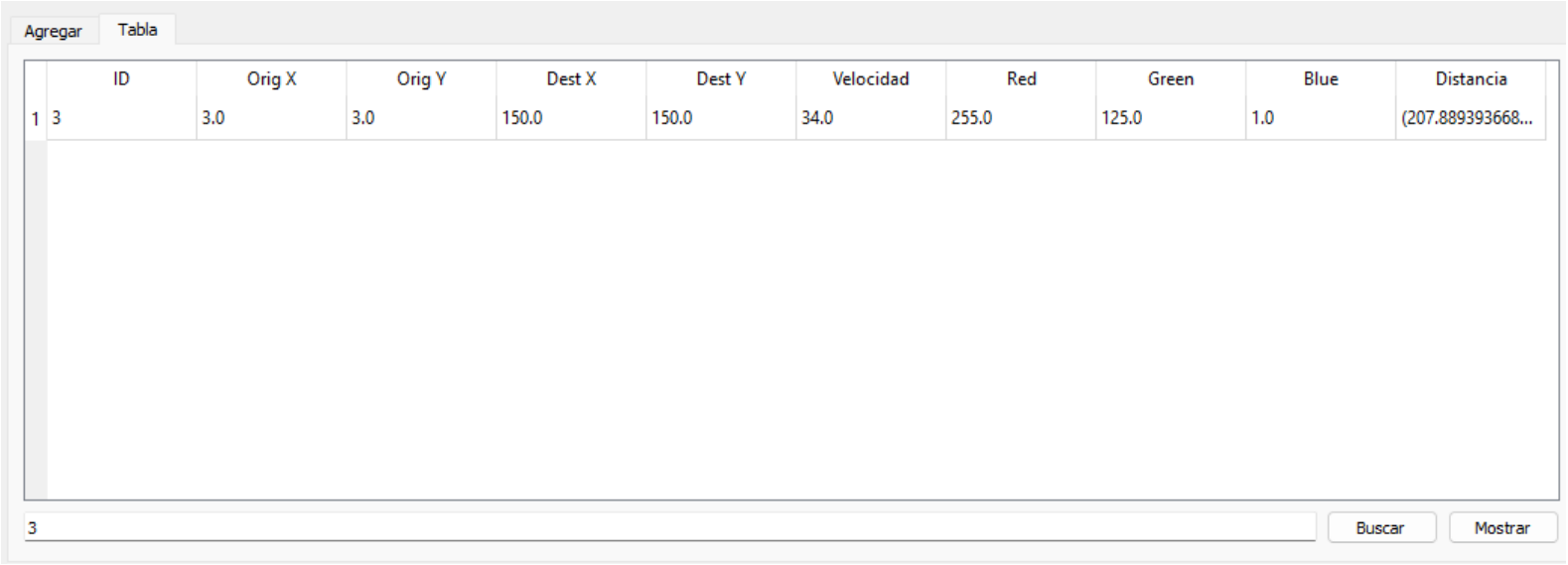
Ahí podemos ver todos los registros anteriormente ingresados en el archivo guardado.json

Lo siguiente a realizar es la búsqueda de un dato ingresando el identificador de la partícula.

Para esto nos dirigimos a la sección de ID de partícula en la parte baja de la interfaz, e ingresamos un ID que concuerde con uno de los 5 ID´s dentro de nuestra tabla anteriormente vista.

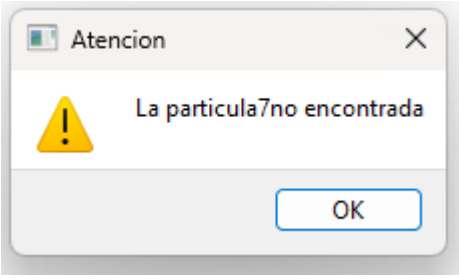
En este caso vamos a buscar el elemento que tenga como identificador el numero 3.

Por lo que procederé a mostrar resultado de esta prueba.



En la parte de debajo podemos ver que ingresamos el numero 3, y arriba observamos que efectivamente el objeto con el numero de identificador 3, se muestra en esta tabla.

Ahora por último lo que tenemos que mostrar es lo que sucede al momento de ingresar un identificador que no se encuentre en los registros de nuestra tabla, para esto vamos a ingresar el número 7.



Este es el mensaje que nos arroja el programa cuando no concuerda el número de identificación que nosotros ingresamos, con aquel que se encuentra dentro de la tabla.

Conclusiones

Primero que nada, al momento de realizar la actividad, no tuve ningún tipo de problema, esto porque la información que se nos mostraba en el video me ayudo a realizar la actividad de una forma bastante fácil.

Considero que el conocer como es que podemos utilizar los diferentes tipos de formatos para mostrar información nos puede beneficiar para tener una mejor organización al momento de nosotros poder visualizar dicha información.

Además, conocer las diferentes herramientas que nosotros podemos tener con el uso de QT, nos abren muchas puertas al momento de diseñar una interfaz intuitiva y con esto poder hacer un mejor uso y adaptar estas herramientas a lo que nosotros deseamos realizar como objetivo.

Referencias

https://www.youtube.com/watch?v=1yEpAHaiMxs&t=487s – Michel Davalos Boites - Pyside2 – QTableWidget (Qt for Python) (V)

Código.

Para el código, igual que las veces pasadas tenemos 5 archivos los cuales los agregare de esta forma.

Userinterface.py

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

#Se crea la aplicacion de QT
app = QApplication()

#Se crea un botton con la palabra hola
window = MainWindow()

#Se hace visible el boton.
window.show()

#QT Loop
sys.exit(app.exec_())
```

ui_interfaz.py

```
from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.objectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.resize(615, 453)
        self.actionAbrir = QAction(MainWindow)
        self.actionAbrir.setObjectName(u"actionAbrir")
        self.actionGuardar = QAction(MainWindow)
        self.actionGuardar.setObjectName(u"actionGuardar")
        self.centralwidget = QWidget(MainWindow)
        self.centralwidget.setObjectName(u"centralwidget")
        self.gridLayout_4 = QGridLayout(self.centralwidget)
        self.gridLayout_4.setObjectName(u"gridLayout_4")
        self.tabWidget = QTabWidget(self.centralwidget)
        self.tabWidget.setObjectName(u"tabWidget")
        self.tab = QWidget()
        self.tab.setObjectName(u"tab")
        self.gridLayout_2 = QGridLayout(self.tab)
        self.gridLayout_2.setObjectName(u"gridLayout_2")
        self.groupBox = QGroupBox(self.tab)
        self.groupBox.setObjectName(u"groupBox")
        self.gridLayout = QGridLayout(self.groupBox)
        self.gridLayout.setObjectName(u"gridLayout")
        self.Green_lineEdit = QLineEdit(self.groupBox)
        self.Green_lineEdit.setObjectName(u"Green_lineEdit")

        self.gridLayout.addWidget(self.Green_lineEdit, 12, 3, 1, 2)

        self.Origen_Y_lineEdit = QLineEdit(self.groupBox)
        self.Origen_Y_lineEdit.setObjectName(u"Origen_Y_lineEdit")

        self.gridLayout.addWidget(self.Origen_Y_lineEdit, 1, 3, 1, 2)

        self.DestinoX = QLabel(self.groupBox)
        self.DestinoX.setObjectName(u"DestinoX")

        self.gridLayout.addWidget(self.DestinoX, 2, 1, 1, 2)
```

```
        self.DestinoX_lineEdit = QLineEdit(self.groupBox)
        self.DestinoX_lineEdit.setObjectName(u"DestinoX_lineEdit")

        self.gridLayout.addWidget(self.DestinoX_lineEdit, 2, 3, 1, 2)

        self.Blue = QLabel(self.groupBox)
        self.Blue.setObjectName(u"Blue")

        self.gridLayout.addWidget(self.Blue, 14, 1, 1, 1)

        self.Red_lineEdit = QLineEdit(self.groupBox)
        self.Red_lineEdit.setObjectName(u"Red_lineEdit")

        self.gridLayout.addWidget(self.Red_lineEdit, 8, 3, 1, 2)

        self.Green = QLabel(self.groupBox)
        self.Green.setObjectName(u"Green")

        self.gridLayout.addWidget(self.Green, 12, 1, 1, 1)

        self.Velocidad_lineEdit = QLineEdit(self.groupBox)
        self.Velocidad_lineEdit.setObjectName(u"Velocidad_lineEdit")

        self.gridLayout.addWidget(self.Velocidad_lineEdit, 4, 3, 1, 2)

        self.DestinoY_lineEdit = QLineEdit(self.groupBox)
        self.DestinoY_lineEdit.setObjectName(u"DestinoY_lineEdit")

        self.gridLayout.addWidget(self.DestinoY_lineEdit, 3, 3, 1, 2)

        self.Velocidad = QLabel(self.groupBox)
        self.Velocidad.setObjectName(u"Velocidad")

        self.gridLayout.addWidget(self.Velocidad, 4, 1, 1, 1)

        self.Origen_X_lineEdit = QLineEdit(self.groupBox)
        self.Origen_X_lineEdit.setObjectName(u"Origen_X_lineEdit")

        self.gridLayout.addWidget(self.Origen_X_lineEdit, 0, 3, 1, 2)

        self.Origen_Y = QLabel(self.groupBox)
        self.Origen_Y.setObjectName(u"Origen_Y")

        self.gridLayout.addWidget(self.Origen_Y, 1, 1, 1, 1)
```

```
        self.Red = QLabel(self.groupBox)
        self.Red.setObjectName(u"Red")

        self.gridLayout.addWidget(self.Red, 8, 1, 1, 1)

        self.DestinoY = QLabel(self.groupBox)
        self.DestinoY.setObjectName(u"DestinoY")

        self.gridLayout.addWidget(self.DestinoY, 3, 1, 1, 2)

        self.Blue_lineEdit = QLineEdit(self.groupBox)
        self.Blue_lineEdit.setObjectName(u"Blue_lineEdit")

        self.gridLayout.addWidget(self.Blue_lineEdit, 14, 3, 1, 2)

        self.Origen_X = QLabel(self.groupBox)
        self.Origen_X.setObjectName(u"Origen_X")

        self.gridLayout.addWidget(self.Origen_X, 0, 1, 1, 1)

        self.Agregar_inicio_pushButton = QPushButton(self.groupBox)
        self.Agregar_inicio_pushButton.setObjectName(u"Agregar_inicio_pushBu
tton")

        self.gridLayout.addWidget(self.Agregar_inicio_pushButton, 16, 1, 1,
4)

        self.Agregar_final_pushButton = QPushButton(self.groupBox)
        self.Agregar_final_pushButton.setObjectName(u"Agregar_final_pushButt
on")

        self.gridLayout.addWidget(self.Agregar_final_pushButton, 17, 1, 1,
4)

        self.Mostrar_pushButton = QPushButton(self.groupBox)
        self.Mostrar_pushButton.setObjectName(u"Mostrar_pushButton")

        self.gridLayout.addWidget(self.Mostrar_pushButton, 18, 1, 1, 4)


        self.gridLayout_2.addWidget(self.groupBox, 0, 0, 1, 1)

        self.Salida = QPlainTextEdit(self.tab)
        self.Salida.setObjectName(u"Salida")
```

```
        self.gridLayout_2.addWidget(self.Salida, 0, 1, 1, 1)

        self.tabWidget.addTab(self.tab, "")
        self.tab_2 = QWidget()
        self.tab_2.setObjectName(u"tab_2")
        self.gridLayout_3 = QGridLayout(self.tab_2)
        self.gridLayout_3.setObjectName(u"gridLayout_3")
        self.buscar_lineEdit = QLineEdit(self.tab_2)
        self.buscar_lineEdit.setObjectName(u"buscar_lineEdit")

        self.gridLayout_3.addWidget(self.buscar_lineEdit, 1, 0, 1, 1)

        self.mostrar_tabla_pushButton = QPushButton(self.tab_2)
        self.mostrar_tabla_pushButton.setObjectName(u"mostrar_tabla_pushButt
on")

        self.gridLayout_3.addWidget(self.mostrar_tabla_pushButton, 1, 2, 1,
1)

        self.buscar_pushButton = QPushButton(self.tab_2)
        self.buscar_pushButton.setObjectName(u"buscar_pushButton")

        self.gridLayout_3.addWidget(self.buscar_pushButton, 1, 1, 1, 1)

        self.tabla = QTableWidget(self.tab_2)
        self.tabla.setObjectName(u"tabla")

        self.gridLayout_3.addWidget(self.tabla, 0, 0, 1, 3)

        self.tabWidget.addTab(self.tab_2, "")

        self.gridLayout_4.addWidget(self.tabWidget, 1, 0, 1, 1)

        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QMenuBar(MainWindow)
        self.menubar.setObjectName(u"menubar")
        self.menubar.setGeometry(QRect(0, 0, 615, 22))
        self.menuArchivo = QMenu(self.menubar)
        self.menuArchivo.setObjectName(u"menuArchivo")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QStatusBar(MainWindow)
        self.statusbar.setObjectName(u"statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.menubar.addAction(self.menuArchivo.menuAction())
```

```
        self.menuArchivo.addAction(self.actionAbrir)
        self.menuArchivo.addAction(self.actionGuardar)

        self.retranslateUi(MainWindow)

        self.tabWidget.setCurrentIndex(1)


        QMetaObject.connectSlotsByName(MainWindow)
    # setupUi

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(QCoreApplication.translate("MainWindow",
u"MainWindow", None))
        self.actionAbrir.setText(QCoreApplication.translate("MainWindow",
u"Abrir", None))
#if QT_CONFIG(shortcut)
        self.actionAbrir.setShortcut(QCoreApplication.translate("MainWindow"
, u"Ctrl+O", None))
#endif // QT_CONFIG(shortcut)
        self.actionGuardar.setText(QCoreApplication.translate("MainWindow",
u"Guardar", None))
#if QT_CONFIG(shortcut)
        self.actionGuardar.setShortcut(QCoreApplication.translate("MainWindo
w", u"Ctrl+S", None))
#endif // QT_CONFIG(shortcut)
        self.groupBox.setTitle(QCoreApplication.translate("MainWindow",
u"Particula", None))
        self.DestinoX.setText(QCoreApplication.translate("MainWindow",
u"Destino X: (0 - 500) ", None))
        self.Blue.setText(QCoreApplication.translate("MainWindow", u"Blue:
(0 - 255)", None))
        self.Green.setText(QCoreApplication.translate("MainWindow", u"Green:
(0 - 255)", None))
        self.Velocidad.setText(QCoreApplication.translate("MainWindow",
u"Velocidad:", None))
        self.Origen_Y.setText(QCoreApplication.translate("MainWindow",
u"Origen Y: (0 - 500)", None))
        self.Red.setText(QCoreApplication.translate("MainWindow", u"Red: (0
- 255)", None))
        self.DestinoY.setText(QCoreApplication.translate("MainWindow",
u"Destino Y: (0 - 500) ", None))
        self.Origen_X.setText(QCoreApplication.translate("MainWindow",
u"Origen X: (0 - 500)", None))
```

```
        self.Agregar_inicio_pushButton.setText(QCoreApplication.translate("M
ainWindow", u"Agregar inicio", None))
        self.Agregar_final_pushButton.setText(QCoreApplication.translate("Ma
inWindow", u"Agregar Final", None))
        self.Mostrar_pushButton.setText(QCoreApplication.translate("MainWind
ow", u"Mostrar elementos", None))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab),
QCoreApplication.translate("MainWindow", u"Agregar", None))
        self.buscar_lineEdit.setPlaceholderText(QCoreApplication.translate("
MainWindow", u"ID de la particula.", None))
        self.mostrar_tabla_pushButton.setText(QCoreApplication.translate("Ma
inWindow", u"Mostrar", None))
        self.buscar_pushButton.setText(QCoreApplication.translate("MainWindo
w", u"Buscar", None))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2),
QCoreApplication.translate("MainWindow", u"Tabla", None))
        self.menuArchivo.setTitle(QCoreApplication.translate("MainWindow",
u"Archivo", None))
    # retranslateUi
```

Particula.py

```
from cmath import sqrt
import math

class Particula:
    def __init__(self, id = 0.0, origen_X= 0.0, origen_Y= 0.0,
                 destino_X= 0.0, destino_Y= 0.0, velocidad= 0.0,
                 red= 0.0, green= 0.0, blue= 0.0):
        self.__id = str(id)
        self.__origen_x = str(origen_X)
        self.__origen_y = str(origen_Y)
        self.__destino_x = str(destino_X)
        self.__destino_y = str(destino_Y)
        self.__velocidad = str(velocidad)
        self.__red = str(red)
        self.__green = str(green)
        self.__blue = str(blue)
        #self.__distancia = (distancia_euclidiana(origen_X, origen_Y,
destino_X, destino_Y))

    def __str__(self):
        return(
```

```
            'ID: ' + self.__id + '\n' +
            'Origen X: ' + self.__origen_x + '\n' +
            'Origen Y: ' + self.__origen_y + '\n' +
            'Destino X: ' + self.__destino_x + '\n' +
            'Destino Y: ' + self.__destino_y + '\n' +
            'Velocidad: ' + self.__velocidad + ' M/S \n' +
            'Rojo: ' + self.__red + '\n' +
            'Verde: ' + self.__green + '\n' +
            'Azul: ' + self.__blue + '\n'
            #'Distancia: ' + self.__distancia +  ' M \n'
            )

    @property
    def id(self):
        return self.__id

    @property
    def origenX(self):
        return self.__origen_x

    @property
    def origenY(self):
        return self.__origen_y

    @property
    def destinoX(self):
        return self.__destino_x

    @property
    def destinoY(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):
        ox = float(self.__origen_x)
        oy = float(self.__origen_y)
        dx = float(self.__destino_x)
        dy = float(self.__destino_y)
        return repr(sqrt(pow((ox - dx), 2) + pow((oy - dy), 2)))

    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
        }
```

Mainwindow.py

```
from operator import truediv
from re import L
from wsgiref import headers
from PySide2.QtWidgets import QMainWindow, QFileDialog, QMessageBox,
QTableWidgetItem
from PySide2.QtCore import Slot
from ui_interfaz import Ui_MainWindow
from libreria import Lista
from particula import Particula

class MainWindow(QMainWindow):

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

        self.lista = Lista()
```

```
        self.id = int(0)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.Agregar_inicio_pushButton.clicked.connect(self.click_agregar
_inicio)
        self.ui.Agregar_final_pushButton.clicked.connect(self.click_agregar_
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_elemento)

    @Slot()
    def mostrar_tabla(self):
        self.ui.tabla.setColumnCount(10)
        headers = ["ID","Orig X", "Orig Y", "Dest X", "Dest Y", "Velocidad"
,"Red",  "Green", "Blue", "Distancia"]
        self.ui.tabla.setHorizontalHeaderLabels(headers)

        self.ui.tabla.setRowCount(len(self.lista))

        row = 0
        for particula in self.lista:
            id_widget = QTableWidgetItem(particula.id)
            origenX_widget = QTableWidgetItem(particula.origenX)
            origenY_widget = QTableWidgetItem(particula.origenY)
            destinoX_widget = QTableWidgetItem(particula.destinoX)
            destinoY_widget = QTableWidgetItem(particula.destinoY)
            velocidad_widget = QTableWidgetItem(particula.velocidad)
            red_widget = QTableWidgetItem(particula.red)
            green_widget = QTableWidgetItem(particula.green)
            blue_widget = QTableWidgetItem(particula.blue)
            distancia_widget = QTableWidgetItem(particula.distancia)

            self.ui.tabla.setItem(row, 0, id_widget)
            self.ui.tabla.setItem(row, 1, origenX_widget)
            self.ui.tabla.setItem(row, 2, origenY_widget)
            self.ui.tabla.setItem(row, 3, destinoX_widget)
            self.ui.tabla.setItem(row, 4, destinoY_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 buscar_elemento(self):
        identificador = self.ui.buscar_lineEdit.text()

        find = False
        for particula in self.lista:
            if identificador == particula.id:
                self.ui.tabla.clear()
                self.ui.tabla.setColumnCount(10)
                headers = ["ID","Orig X", "Orig Y", "Dest X", "Dest Y",
                "Velocidad" ,"Red",  "Green", "Blue", "Distancia"]
                self.ui.tabla.setHorizontalHeaderLabels(headers)
                self.ui.tabla.setRowCount(1)

                id_widget = QTableWidgetItem(particula.id)
                origenX_widget = QTableWidgetItem(particula.origenX)
                origenY_widget = QTableWidgetItem(particula.origenY)
                destinoX_widget = QTableWidgetItem(particula.destinoX)
                destinoY_widget = QTableWidgetItem(particula.destinoY)
                velocidad_widget = QTableWidgetItem(particula.velocidad)
                red_widget = QTableWidgetItem(particula.red)
                green_widget = QTableWidgetItem(particula.green)
                blue_widget = QTableWidgetItem(particula.blue)
                distancia_widget = QTableWidgetItem(particula.distancia)

                self.ui.tabla.setItem(0, 0, id_widget)
                self.ui.tabla.setItem(0, 1, origenX_widget)
                self.ui.tabla.setItem(0, 2, origenY_widget)
                self.ui.tabla.setItem(0, 3, destinoX_widget)
                self.ui.tabla.setItem(0, 4, destinoY_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)

                find = True
                return

        if not find:
```

```
            QMessageBox.warning(
                self,
                "Atencion",
                "La particula" + identificador + "no encontrada"
            )



    @Slot()
    def action_abrir_archivo(self):
        dir = QFileDialog.getOpenFileName(
            self,
            "Abrir Archivo:",
            ".",
            "JSON (*.json)"
        )[0]
        if self.lista.abrir(dir):
            QMessageBox.information(
                self,
                "Exito",
                "Se abrió el archivo " + dir
            )
        else:
            QMessageBox.critical(
                self,
                "Error",
                "Error al abrir el archivo " + dir
            )

    @Slot()
    def action_guardar_archivo(self):
        #print("Guardando archivo")
        dir = QFileDialog.getSaveFileName(
            self,
            "Guardar como:",
            ".",
            "JSON (*.json)"
        )[0]

        if self.lista.guardar(dir):
            QMessageBox.information(
                self,
                "Éxito",
                "Se pudo crear y guardar datos del archivo " + dir
            )
```

```
        else:
            QMessageBox.critical(
                self,
                "Error",
                "No se pudo crear y/o guardar datos en el archivo " + dir
            )

    @Slot()
    def click_agregar_inicio(self):
        self.id = self.id + int(1)
        origenx = float(self.ui.Origen_X_lineEdit.text())
        origeny = float(self.ui.Origen_Y_lineEdit.text())
        destinox = float(self.ui.DestinoX_lineEdit.text())
        destinoy = float(self.ui.DestinoY_lineEdit.text())
        velocidad = float(self.ui.Velocidad_lineEdit.text())
        red = float(self.ui.Red_lineEdit.text())
        green = float(self.ui.Green_lineEdit.text())
        blue = float(self.ui.Blue_lineEdit.text())

        particula = Particula(self.id, origenx, origeny, destinox, destinoy,
velocidad, red, green, blue)

        self.lista.agregar_inicio(particula)

    @Slot()
    def click_agregar_final(self):
        self.id = self.id + int(1)
        origenx = float(self.ui.Origen_X_lineEdit.text())
        origeny = float(self.ui.Origen_Y_lineEdit.text())
        destinox = float(self.ui.DestinoX_lineEdit.text())
        destinoy = float(self.ui.DestinoY_lineEdit.text())
        velocidad = float(self.ui.Velocidad_lineEdit.text())
        red = float(self.ui.Red_lineEdit.text())
        green = float(self.ui.Green_lineEdit.text())
        blue = float(self.ui.Blue_lineEdit.text())

        particula = Particula(self.id, origenx, origeny, destinox, destinoy,
velocidad, red, green, blue)

        self.lista.agregar_final(particula)

        #self.ui.Salida.insertPlainText()

    @Slot()
    def click_mostrar(self):
```

```
        self.ui.Salida.clear()
        self.ui.Salida.insertPlainText(str(self.lista))
```

libreria.py

```
from particula import Particula
import json

class Lista:
    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 __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, direccion):
        try:
            with open(direccion, 'w') as archivo:
                lista = [ particula.to_dict() for particula in
self.__particulas ]
                json.dump(lista, archivo, indent=5)
            return 1
        except:
            return 0

    def abrir(self, direccion):
        try:
            with open(direccion, 'r') as archivo:
                lista = json.load(archivo)
                self.__particulas = [Particula(**particula) for particula in
lista]
            return 1
        except:
            return 0
```