Actividad 08 - QTableWidget

JAIRO CAIN SANCHEZ ESTRADA// Luis Angel Elisea Graciano

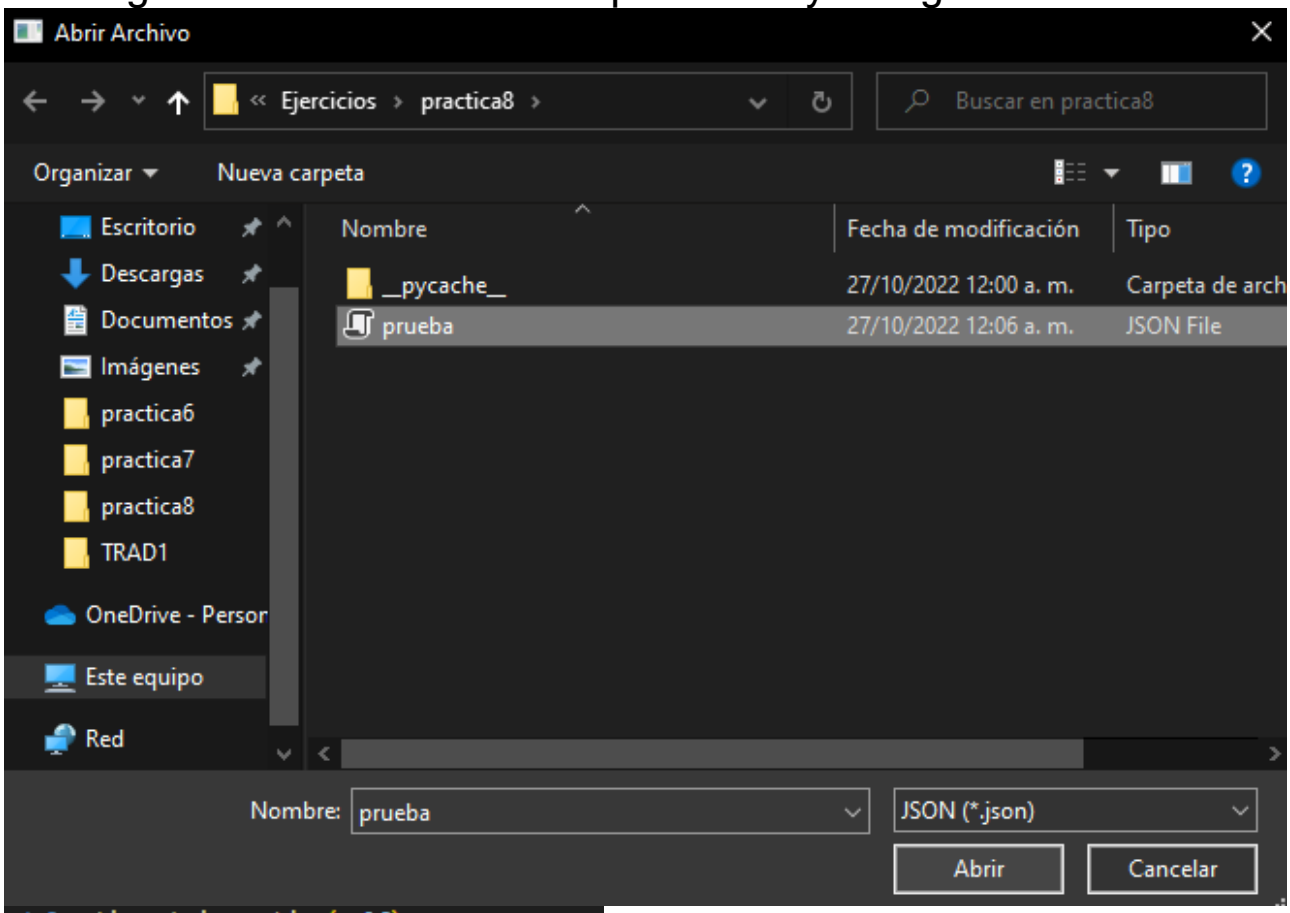
SEMINARIO DE SOLUCION DE PROBLEMAS DE ALGORITMIA

Lineamientos de evaluación

- Se agrego las opciones de mostrar las partículas en una tabla desde la interfaz gráfica y buscar
- Se agrego las alertas para cuando no se encuentra una partícula cuando se busca por su índice

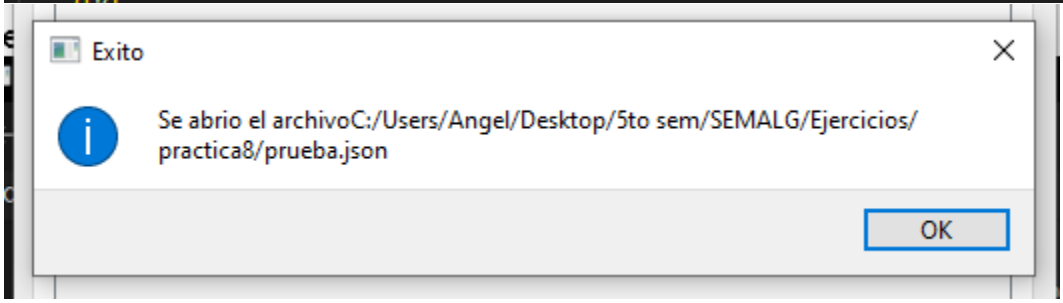
Desarrollo

Se carga un archivo .JSON con partículas ya cargadas

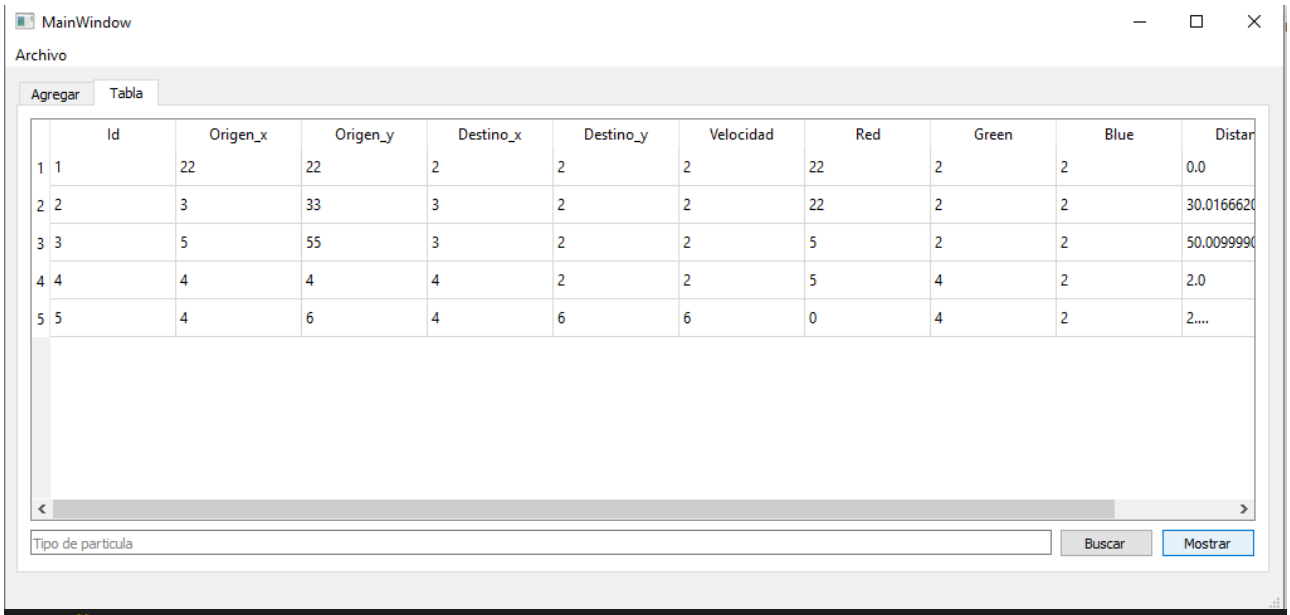



```
def action_abrir_archivo(self):
    #print("abrir")
    ubicacion=QFileDialog.getOpenFileName(
        self,
        "Abrir Archivo",
        ".",
        "JSON (*.json)"
    )[0]
```

```
if self.particulas.abrir(ubicacion):
    QMessageBox.information(
        self,
        "Exito",
        "Se abrio el archivo" + ubicacion
    )
```



Posterior mente de cargar el archivo .JSON se muestran las particulas en la tabla creada.




```python
@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.particulas))
    row=0
    for particula in self.particulas:
        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.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.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
```

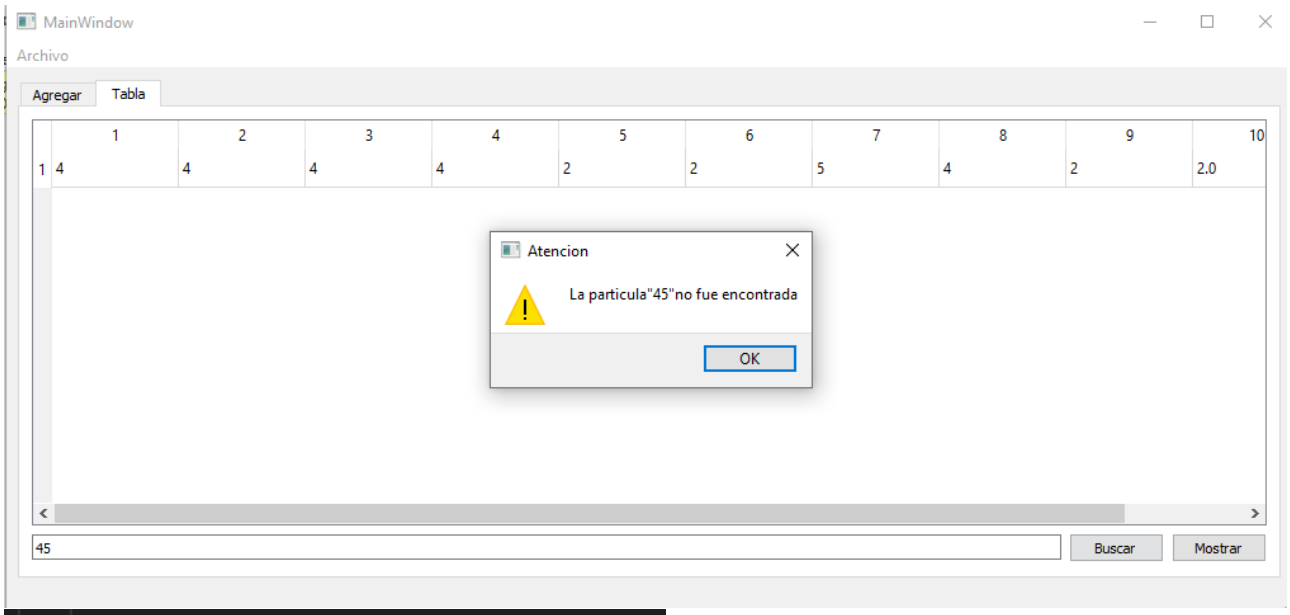
Se realiza una búsqueda de la partícula con el id 4:


```
for particula in self.particulas:
    if id==particula.id:
        self.ui.tabla.clear()
        self.ui.tabla.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.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.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
```

Se realiza una búsqueda de la partícula con el id 45:




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

Conclusiones

Esta práctica me sirvió bastante ya que necesito crear una interfaz con estas características y la manera de implementar esta tabla y la forma de mostrar la información me ayudo bastante.

Referencias

Boites, M. D. [UC3jaAJ6X3vMZJKpi9KAcY0w]. (2020c, October 29). PySide2 - QTableWidget (Qt for Python)(V). Youtube. https://www.youtube.com/watch?v=1yEpAHaiMxs

Código 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,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, destino_x, origen_y,
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_dic(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,
            #'distancia':self.__distancia

        }
```

Algoritmos.py

```python
from math import sqrt
def distancia_euclidiana(x_1, y_1, x_2, y_2):
    """ Calcula la distancia euclidiana
    Devuelve el resultado de la fórmula
    También se le conoce a la fórmula como:
    distancia entre dos puntos
    Parámetros:
    x_1 -- origen_x
    y_1 -- origen_y
    x_2 -- destino_x
    y_2 -- destino_y
    """
    return(sqrt(((x_2 - x_1)**2) + ((y_2 - y_1)**2)))
```

particulas.py

```python
from particula import Particula
import json
class Particulas:
    def __init__(self):
        self.__particulas = []

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

    def agregar_final(self,particula:Particula):
        self.__particulas.append(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,ubicacion):
        try:
            with open(ubicacion,"w") as archivo:
                lista=[particula.to_dic() 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
```

mainwindow.py

```
from PySide2.QtWidgets import QMainWindow, QFileDialog,
QMessageBox,QTableWidgetItem
from PySide2.QtCore import Slot
from ui_mainwindow import Ui_MainWindow
from particula import Particula
from particulas import Particulas

#pyside2-uic mainwindow.ui para pasar de .ui a python
class MainWindow(QMainWindow):
    def __init__(self):
```

```
        super(MainWindow, self).__init__()

        self.particulas= Particulas()

        self.ui=Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.Agregar_final_pushButton.clicked.connect(self.click_agregar)
        self.ui.Agregar_inicio_pushButton.clicked.connect(self.click_agregar_inicio
)
        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.particulas:
            if id==particula.id:
                self.ui.tabla.clear()
                self.ui.tabla.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.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.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"{id}"no fue encontrada'
            )

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

        self.ui.tabla.setRowCount(len(self.particulas))
        row=0
        for particula in self.particulas:
            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.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.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):
        #print("abrir")
        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",
                "No se pudo abrir el archivo"
            )
    @Slot()
    def action_guardar_archivo(self):
        #print("guardar")
        ubicacion=QFileDialog.getSaveFileName(
            self,
            "Guardar Archivo",
            ".",
            "JSON (*.json)"

        )[0]
        print(ubicacion)
        if self.particulas.guardar(ubicacion):
            QMessageBox.information(
                self,
                "Exito",
                "Se pudo crear el archivo" + ubicacion
            )
        else:
            QMessageBox.critical(
                self,
                "Error",
                "No se pudo crear el archivo"
            )
    @Slot()
    def click_mostrar(self):
        self.ui.salida.clear()
        self.ui.salida.insertPlainText(str(self.particulas))

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

```
        id=self.ui.Id_spinBox.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()

        Particula1=Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,red
,green,blue)
        self.particulas.agregar_final(Particula1)

    @Slot()
    def click_agregar_inicio(self):
        id=self.ui.Id_spinBox.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()

        Particula1=Particula(id,origen_x,origen_y,destino_x,destino_y,velocidad,red
,green,blue)
        self.particulas.agregar_inicio(Particula1)
```

Main.py

```
from PySide2.QtWidgets import QApplication
from  mainwindow import MainWindow
#pyside2-uic mainwindow.ui para pasar de .ui a python
#pyside2-uic mainwindow.ui >ui_mainwindow.py
import sys
app=QApplication()
window=MainWindow()
window.show()
sys.exit(app.exec_())
```