Seminario de Solución de Problemas de Algoritmia
Becerra Ruan Juan Martin
220286911

Actividad 4. QPlainTextEdit

Introducción

Para esta actividad unificaras la clase *Neurona*, la clase administradora y UI (User Interface). Esto para poder obtener la información de la GUI, crear el objeto de la clase *Neurona* y agregarlo a la clase administradora.

Posteriormente, la clase administradora se encargará de clasificar las *Neuronas* por voltaje, así como de generar un grafo con las *Neuronas* que administra para implementar los algoritmos sobre grafos.

Todo lo anterior será controlado por la interfaz, usando botones, acciones y diferentes widgets para visualizar información. Esto se le conoce como MVC (model-view-controller), el cual es un patrón de diseño para el desarrollo de software para implementar interfaces graficas.

Al usar c++ con Qt, se usa MVC (model-view-controller), en donde a model se refiera a la definición de las clases y objetos, la view se refiere a la interfaz de usuario (UI) diseñada en el Designer y el controller se refiere a él mainwindow.cpp que se conecta tanto a la interfaz de usuario como a la clase administradora.

Ten en cuenta que las próximas actividades necesitaran de las actividades previas, ya que se ira´ construyendo la interfaz gráfica de usuario usando diferentes widgets (componentes gráficos de Qt) para diferentes fines e incluso para ejecutar algunos algoritmos usando ciertos widgets.

Objetivo

El alumno complementara el desarrollo de la GUI obteniendo la información de los widgets QLineEdit y QSpinBox, así como hacer uso del widget QPlainTextEdit para mostrar información hacia la interfaz gráfica.

Seminario de Solución de Problemas de Algoritmia
Becerra Ruan Juan Martin
220286911

Instrucciones

1. Modifica el proyecto de la interfaz gráfica desarrollada en la actividad anterior para que sea capaz de:

- Agregar un objeto de tipo *Neurona* a la clase administradora de *Neuronas* desde la interfaz gráfica de usuario, usando los métodos:
  - agregar inicio(): Usar un botón para insertar un objeto de tipo *Neurona* al inicio de la lista de *Neuronas*.
  - agregar final(): Usar un botón para insertar un objeto de tipo *Neurona* al final de la lista de *Neuronas*.
- Mostrar en un QPlanTextEdit en la interfaz gráfica la información de todas las *Neuronas* que contiene la clase administradora. Con lo anterior, es necesario agregar un botón que hará se visualice en el QPlanTextEdit toda la información de las *Neuronas* del administrador de *Neuronas*.

Desarrollo

Al realizar que la interfaz grafica logre guardar las neuronas en la lista fue necesario utilizar los botones que fueron programados. Para se generó el código de los tres botones.

- Insertar al inicio
  Primero se guardan los valores de los spinBox en variables. Para después continuar con una restricción que nos permite que no digite 0 ni NULL para que no surjan problemas al registrar una neurona. Si son diferentes a 0 y NULL, se crea una nueva neurona con los valores de la clase neurona y se agregan al inicio de la lista. Por último, se usa una función de clear() que sirve para limpiar los spinBox.

```
void MainWindow::on_pushButton_clicked(){
    //INSERTAR AL INICIO
    int id, posicion_x,posicion_y,red,green,blue;
    double voltaje;
    id=ui->id_spinBox->value();
    voltaje=ui->voltaje_doubleSpinBox->value();
    posicion_x=ui->Pos_x_spinBox_2->value();
    posicion_y=ui->Pos_y_spinBox_2->value();
    red=ui->red_spinBox_4->value();
    green=ui->green_spinBox_5->value();
    blue=ui->spinBox_6->value();
    if(id == 0 or voltaje == 0 or id == NULL or voltaje
== NULL){
        QMessageBox::warning(this,"Mensaje","Tiene que
existir un voltaje y un id.");
    }else{
        Neurona *nuevo = new
Neurona(id,voltaje,posicion_x,posicion_y,red,green,blue)
;
        lista.push_front(*nuevo);
        ui->id_spinBox->clear();
        ui->voltaje_doubleSpinBox->clear();
        ui->Pos_x_spinBox_2->clear();
        ui->Pos_y_spinBox_2->clear();
        ui->red_spinBox_4->clear();
        ui->green_spinBox_5->clear();
        ui->spinBox_6->clear();
    }
}
```

- Insertar al final
  Esta es igual a la anterior solo cambia que en vez de insertar al inicio, lo inserta en el final.

```
void MainWindow::on_pushButton_2_clicked(){
    //INSERTAR AL FINAL
    int id, posicion_x,posicion_y,red,green,blue;
    double voltaje;
    id=ui->id_spinBox->value();
    voltaje=ui->voltaje_doubleSpinBox->value();
    posicion_x=ui->Pos_x_spinBox_2->value();
    posicion_y=ui->Pos_y_spinBox_2->value();
```

```
    red=ui->red_spinBox_4->value();
    green=ui->green_spinBox_5->value();
    blue=ui->spinBox_6->value();
    if(id == 0 or voltaje == 0 or id == NULL or voltaje
== NULL){
        QMessageBox::warning(this,"Mensaje","Tiene que
existir un voltaje y un id.");
    }else{
        Neurona *nuevo = new
Neurona(id,voltaje,posicion_x,posicion_y,red,green,blue)
;
        lista.push_back(*nuevo);
        ui->id_spinBox->clear();
        ui->voltaje_doubleSpinBox->clear();
        ui->Pos_x_spinBox_2->clear();
        ui->Pos_y_spinBox_2->clear();
        ui->red_spinBox_4->clear();
        ui->green_spinBox_5->clear();
        ui->spinBox_6->clear();
    }
}
```

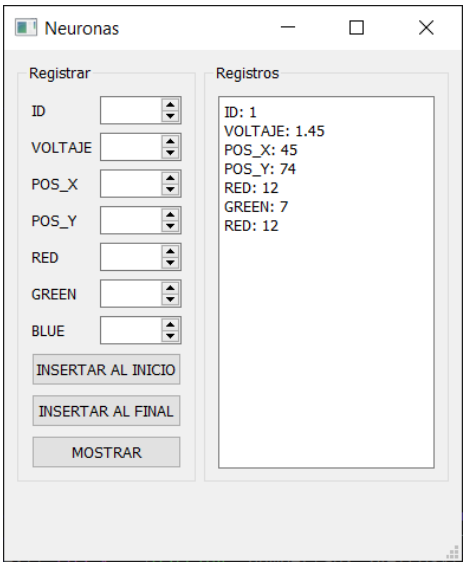
- Mostrar

  Por último el mostrar que primeramente hacemos una condicional que permita saber si la lista esta vacía o no, si eta vacía limpia e imprime en el plainTextEdit “La lista esta vacía…”, y si no hace la misma limpieza del plainTextEdit creamos el iterador de la lista para entrar al bucle while donde inseta las variables de la clase neurona.

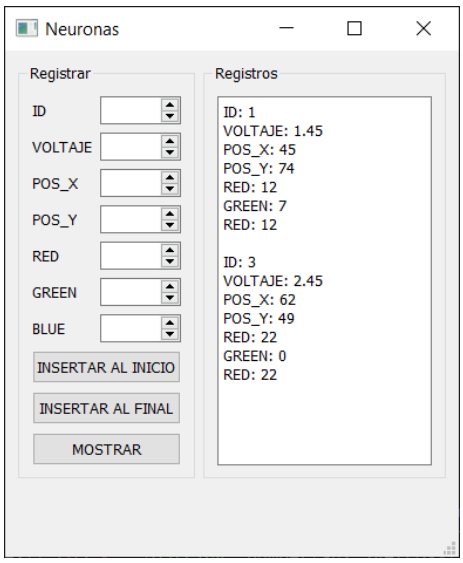
```
void MainWindow::on_pushButton_3_clicked(){
    //MOSTRAR
    if(!lista.empty()){
        ui->plainTextEdit->clear();
        list<Neurona>::iterator aux = lista.begin();
        while(aux!=lista.end()){
            ui->plainTextEdit->insertPlainText("ID:
"+QString::number(aux->getId())+"\n");
            ui->plainTextEdit->insertPlainText("VOLTAJE:
"+QString::number(aux->getVoltaje())+"\n");
```

```
            ui->plainTextEdit->insertPlainText("POS_X:
"+QString::number(aux->getPosX())+"\n");
            ui->plainTextEdit->insertPlainText("POS_Y:
"+QString::number(aux->getPosY())+"\n");
            ui->plainTextEdit->insertPlainText("RED:
"+QString::number(aux->getRed())+"\n");
            ui->plainTextEdit->insertPlainText("GREEN:
"+QString::number(aux->getGreen())+"\n");
            ui->plainTextEdit->insertPlainText("RED:
"+QString::number(aux->getRed())+"\n\n");
            aux++;
        }
    }else{
        ui->plainTextEdit->clear();
        ui->plainTextEdit->insertPlainText("La lista
esta vacia...");
    }
}
```

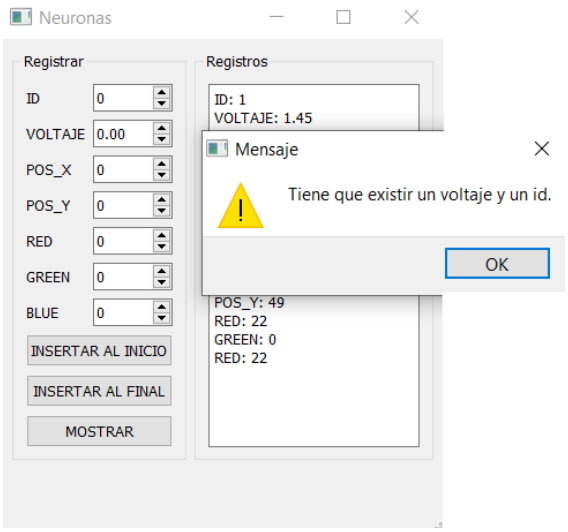
Resultados



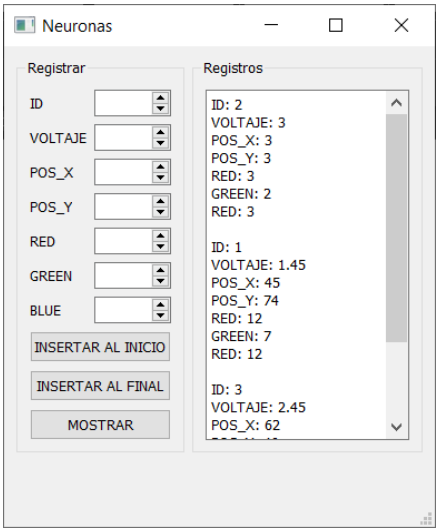
Insertar al inicio



Insertar al final



Cuando intenta insertar un 0 en ID.



Muestra las neuronas y se pasa del cuadro del plainTextEdit usa una barra desplazadora.

Conclusión

Fue fácil entender el funcionamiento de los plainTextEdit pero fue más interesante las configuraciones que puedes realizarle, en mi caso use la de onlyRead que no permite que escriban dentro del plainTextEdit. Cosa que hace que la aplicación sea mas entendible y aprueba de fallas.