

INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria en
Ingeniería y Tecnologías Avanzadas

Programación avanzada

Práctica 5: Interfaces Gráficas de Usuario.
Controles Avanzados

Adalberto Cabrera Vazquez

Boleta: 2023640791

Ingeniería Mecatrónica, Grupo 2MV7

19 de Abril del 2024

1 Objetivo

Crear aplicaciones con interfaces gráficas de usuario (GUI) utilizando los controles avanzados.

2 Introducción

2.1 Ventanas y cuadros de diálogo:

En PyQt, un widget que no está incrustado en un widget principal se llama ventana. Por lo general, las ventanas tienen un marco y una barra de título, aunque también es posible crear ventanas sin dicha decoración.

En Qt, QMainWindow (ventana principal) y las diversas subclasses de QDialog (cuadros de diálogo) son los tipos de ventana más comunes, aunque también está la propia superclase QWidget (clase base de todos los widgets), cuyas instancias sin un padre asignado se convertirán también en ventanas.

2.2 Cuadros de Diálogo:

Los cuadros de diálogo se usan como ventanas secundarias que presentan al usuario opciones y elecciones. Se crean heredando de QDialog y utilizan widgets y diseños para implementar la interfaz de usuario. Qt proporciona una serie de diálogos estándar listos para usar que se pueden usar para tareas estándar como selección de archivos o fuentes.

Evidentemente tanto las ventanas principales como los diálogos se pueden crear con Qt Designer, además es una excelente forma de aprender Qt.

3 Desarrollo

Tomando como base el diagrama de clases que se muestra en la figura 1, diseñe un programa que permita administrar una tienda de mascotas. Su programa deberá permitir administrar el inventario de mascotas, vender mascotas e imprimir reporte de inventario y ventas. Toda la información que se almacene en la aplicación deberá escribirse en un archivo de texto que funcionará como una base de datos. De igual forma, los reportes deberán imprimirse en archivos de texto.

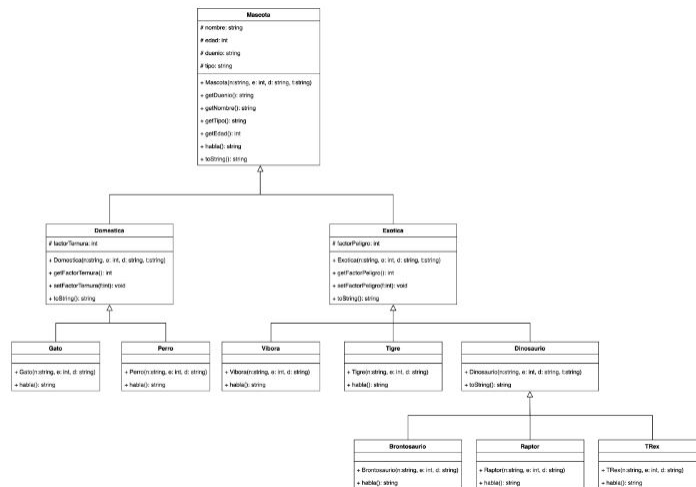


Figure 1: Diagram de clases

3.1 Diseño:

Primero tuve que realizar el diseño de la ventana principal y de los dialogos para cada seccion (administrar/vender/reporte) como se muestra en las siguientes imagenes:

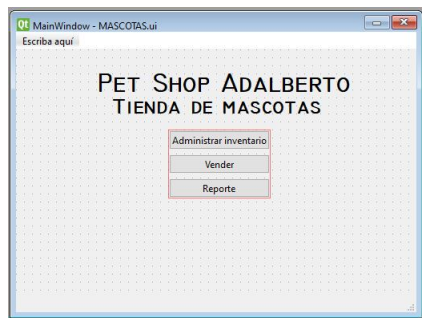


Figure 2: Ventana principal: Mascotas

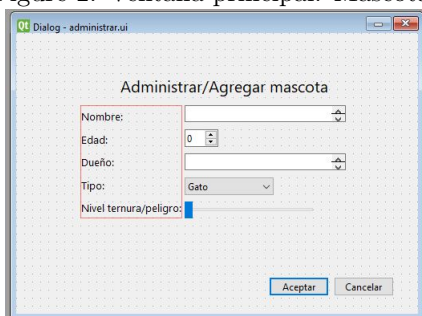


Figure 3: Dialogo administrar

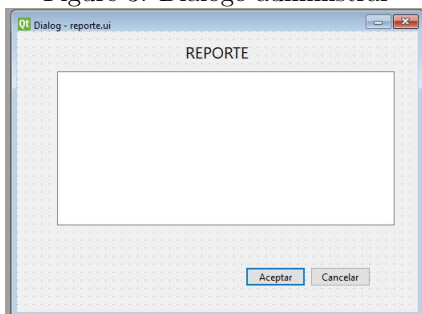


Figure 4: Dialogo reporte

3.2 Código:

Ahora ya teniendo diseñadas nuestra interfaces entonces lo pasamos a codigo usando el Visual Studio Code. En la siguiente imagen se muestran los archivos creados:

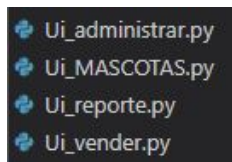


Figure 5: Codigos que se generan automaticamente

Despues reutilice el codigo de clases que ocupamos en la práctica de mascotas haciendole unas modificaciones para en la clase lista se pueda buscar una mascota por su nombre y que se elimine una mascota. De lo demás es lo mismo ya que son las mismas mascotas. En las siguientes imagenes esta

```
'''
PRACTICA 5
05 de Abril del 2024
Cabrera Vazquez Adalberto
2023640791
'''

from abc import ABCMeta, abstractmethod
import numpy as np
import os

#Super clase mascota
class Mascota: ...

class Domestica(Mascota): ...
#sub sub clase que hereda de domestica
class Gato(Domestica): ...
#sub sub clase que hereda de domestica
class Perro(Domestica): ...
#sub clase que hereda de mascota
class Exotica(Mascota): ...
#sub sub clase que hereda de exotica
class Vibora(Exotica): ...
#sub sub clase que hereda de exotica
class Tigre(Exotica): ...
#sub sub clase que hereda de exotica
class Dinosaurio(Exotica): ...
```

Figure 6: Clase mascotas con sus subclases.

la clase lista en donde en la figura 7 se observan agregar mascota y mostrar mascota. En la figura 8 se tienen el buscar mascota por nombre que me servirá para poder venderla y la de eliminar mascota en donde mostrara en el reporte que tal mascota fue vendida. Para eso es necesario acceder al archivo de texto reporte_inventario.txt

```

class Lista:
    def __init__(self) -> None:
        self.__listamascotas = []

    def agregarmascota(self, elemento):
        self.__listamascotas.append(elemento)
        with open('PRACTICAS/reporte_inventario.txt', 'a') as archivo:
            # Escribe información en el archivo

            archivo.write('\n')
            archivo.write('*'*100)
            archivo.write('\n')
            archivo.write('AGREGANDO MASCOTA...\n')
            archivo.write(str(elemento))
            archivo.write('\n')

    def mostrarmascota(self):
        for elemento in self.__listamascotas:
            print('*'*80)
            print(elemento)

```

Figure 7: Agregar y mostrar mascota

```

def buscarnombre(self, nombre):
    for otro in self.__listamascotas:
        if otro.nombre == nombre:
            return otro
    return None

def eliminarcontacto(self, otro):
    with open('PRACTICAS/reporte_inventario.txt', 'a') as archivo:
        # Escribe información en el archivo

        archivo.write('\n')
        archivo.write('*'*100)
        archivo.write('\n')
        archivo.write('MASCOTA VENDIDA: ')
        archivo.write(str(otro))
        archivo.write('\n')

```

Figure 8: Buscar y eliminar mascota

Ya teniendo todo listo, ahora si vamos a crear nuestro programa principal que va a heredar la información de los programas que se generaron automáticamente en python de las ventanas y diálogos. También importamos la información del programa de clases.

```

...
PRACTICA 5
05 de Abril del 2024
Cabrera Vazquez Adalberto
2023640791
...

from PyQt6.QtCore import Qt
from Ui_vender import *
from Ui_administrar import *
from Ui_MASCOTAS import *
from Ui_reporte import *
from PyQt6.QtWidgets import QMainWindow, QDialog, QWidget, QMessageBox, QApplication
import sys
import clases as cd

```

Figure 9: Importamos codigos necesarios

Ahora creamos la lista donde guardemos a las mascotas. Despues creamos la clase MainWindow y dependiendo del boton que apretemos va a iniciar el dialogo que haya puesto. Por ejemplo, si apretamos el boton de administrar va air a la funcion de ShowAdmin y va a ejecutar el dialogo. Y asi con las demás opciones.

```

lista_animales = cd.Lista()

class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self, *parent, **flags) -> None:
        super().__init__(*parent, **flags)
        self.setupUi(self)

        self.btn_administrar.clicked.connect(self.showAdmin)
        self.btn_vender.clicked.connect(self.showVender)
        self.btn_reporte.clicked.connect(self.showReporte)

    def showAdmin(self):
        DialogoAdmin(self, Qt.WindowType.Dialog).exec()

    def showVender(self):
        DialogoVender(self, Qt.WindowType.Dialog).exec()

    def showReporte(self):
        DialogoReporte(self, Qt.WindowType.Dialog).exec()

```

Figure 10: Ventana principal

Ahora muestro lo que se ejecuta cuando apretamos el boton de aceptar que es la figura 11. En esta clase solamente tenemos el boton aceptar pero tenemos varios espacios para meter la información de la mascota a agregar como son el nombre, la edad, el dueño, el tipo de animal y el nivel de ternura o peligro dependiendo el animal y lo guardara cuando apretemos el boton aceptar. La función de guardar simplemente va a leer la información agregada y dependiendo del tipo de animal va a crear al objeto mascota.

```

class DialogoAdmin(QDialog, Ui_admin):
    def __init__(self, parent: QWidget | None = ..., flags: Qt.WindowType = ...) -> None:
        super().__init__(parent, flags)
        self.setupUi(self)
        self.buttonBox.accepted.connect(self.guardar)

    def guardar(self):
        NOMBRE = self.nombre.toPlainText()
        EDAD = self.edad.value()
        DUEÑO = self.dueno.toPlainText()
        TIPO = self.comboBox.currentText()
        NIVEL = self.nivel.value()

        if TIPO=='Gato':
            mascota=cd.Gato(NOMBRE, EDAD, DUEÑO, TIPO, NIVEL)
            lista_animales.agregarmascota(mascota)

        elif TIPO=='Perro':
            mascota=cd.Perro(NOMBRE, EDAD, DUEÑO, TIPO, NIVEL)
            lista_animales.agregarmascota(mascota)

```

Figure 11: Clase Administrar

Para la clase del dialogo de vender tenemos el boton buscar y el boton aceptar. Para poder vender la mascota tenemos que asegurarnos que la mascota esté en el inventario así que primero metemos el nombre de la mascota a buscar y al apretar buscar usara la funcion buscar por nombre y si si está en el inventario entonces te dira que si esta y viceversa. Al presionar aceptar, si esta el animal entonces simplemente se cerrará el dialogo pero si no está te mandara un mensaje de que el animal no existe.

```

class DialogoVender(QDialog, Ui_Vender):
    def __init__(self, parent: QWidget | None = ..., flags: Qt.WindowType = ...) -> None:
        super().__init__(parent, flags)
        self.setupUi(self)
        lista_animales.mostrarmascota()
        self.pushButton.clicked.connect(self.buscar)
        self.buttonBox.accepted.connect(self.venderlo)

    def buscar (self):
        nombrebuscador = self.dueno.toPlainText()
        nombreencontrado = lista_animales.buscarnombre(nombrebuscador)
        if nombreencontrado:
            self.textBrowser.setPlainText("SI SE ENCUENTRA EL ANIMAL EN VENTA")
        else:
            self.textBrowser.setPlainText("NO SE ENCUENTRA EL ANIMAL EN VENTA, BUSQUE OTRO")

    def venderlo(self):
        mascotaenventa=self.textBrowser.toPlainText()
        if mascotaenventa=="NO SE ENCUENTRA EL ANIMAL EN VENTA, BUSQUE OTRO":
            QMessageBox.warning(self, 'Advertencia', 'No puedes vender una mascota que no este en el inventario')
        else:
            nombreborrar = self.dueno.toPlainText()
            lista_animales.eliminarcontacto(nombreborrar)
            lista_animales.mostrarmascota()

```

Figure 12: Dialogo vender

Para la clase Dialogo reporte es que leera lo que hay en el archivo de texto reporte_inventario.txt y lo muestra en el cuadro de texto del dialogo reporte. Asi que cada que se haga un cambio, al presionar reporte, simplemente leera y mostrará en el cuadro de texto.

```
class DialogoReporte(QDialog, Ui_Reporte):
    def __init__(self, parent: QWidget | None = ..., flags: Qt.WindowType = ...) -> None:
        super().__init__(parent, flags)
        self.setupUi(self)
        lista_animales.mostrarmascota()
        '''for elemento in lista_animales:
            self.repositorio.setText('***100)
            self.repositorio.setText(elemento)'''

        with open('PRACTICAS/reporte_inventario.txt', 'r') as f:
            contenido = f.read()
        self.repositorio.setPlainText(contenido)
```

Figure 13: Dialogo reporte

Y para terminar solamente se ejecuta la ventana principal.

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

Figure 14: Ejecucion programa principal

4 Ejecución:

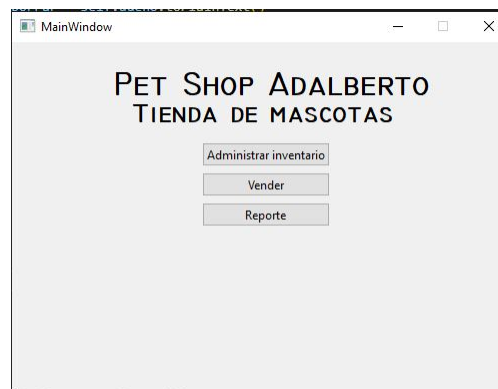


Figure 15: Ventana principal

Dialog

Administrar/Agregar mascota

Nombre:

Edad:

Dueño:

Tipo:

Nivel ternura/peligro:

Figure 16: Agregarndo mascota al inventario

Dialog

Vender mascota

Buscar por nombre:

NO SE ENCUENTRA EL ANIMAL EN VENTA, BUSQUE OTRO

Figure 17: Buscando una mascota que no está en el inventario para vender

Advertencia


 No puedes vender una mascota que no este en el inventario

Figure 18: Ventana de advertencia que no deja vender un animal que no esté en el inventario

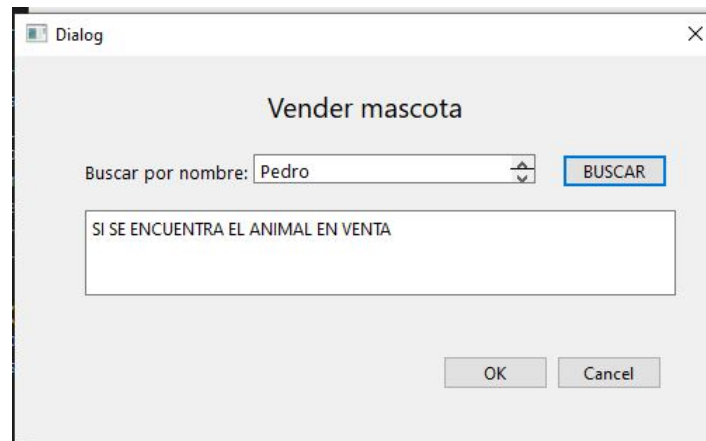


Figure 19: Buscando una mascota que no está en el inventario para vender

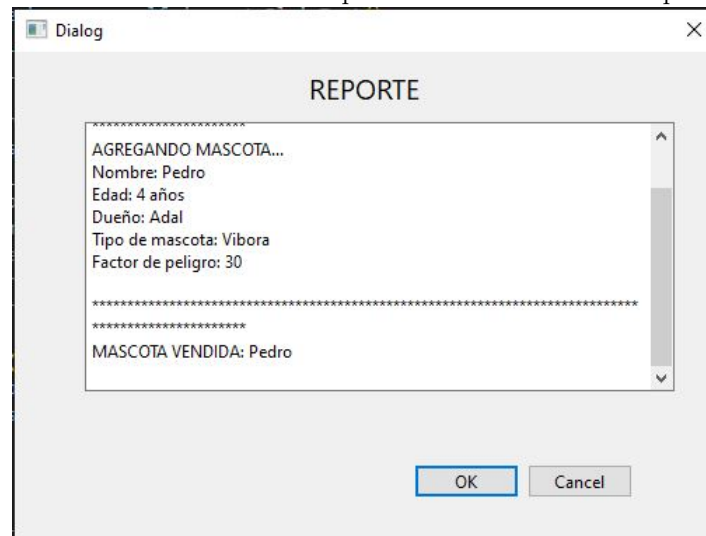


Figure 20: Visualizando el reporte

5 Conclusión

En esta práctica pude conocer y me familiaricé con el lenguaje Python haciendo uso de controles avanzados de PyQtDesigner para poder crear dialogos y administrar informacion que me brindaba el usuario.

6 Bibliografia

- Profe, H. (2018, November 26). Widgets de PyQt 5: Ventanas y cuadros de diálogo - Hektor Profe - Medium. Medium. <https://medium.com/@hektorprofe/widgets-de-pyqt-5-1-ventanas-y-cuadros-de-di%C3%A1logo-937caf847789>