

Задание: и 27.10.23, 3.11.23 4 занятия.

1. Выбрать объект для управления (желательно связанный с тематикой ВКР). Объекты у каждого не должны повторяться. Например джойстик или пульт мобильного, летающего или манипуляционного робота.
2. Продумать наполнение интерфейса. Количество интерактивных объектов не менее 3 (кнопки, уровни и т.п.)
3. Запрограммировать форму интерфейса и оснастить его минимальной графикой (фон, текст и иконки на интерактивных элементах).
4. Связать работу по активации интерактивных элементов с объектом вкр (желательно) или с имитацией логических пробников (световой индикацией подтверждающей действие).
5. Сгенерировать файл проекта в виде пускового файла. И загрузить его в ответ на это задание. Также задание, скриншоты программы разработанного приложения, и листинг кода включить в файл ворд (желательно pdf) и загрузить в ответ на задание.

Ссылка на гит с проектом:

https://github.com/nameunique/DaGOI_lab1/tree/main

Скриншот программы:

form-lab1

Выберите порт:

Обновить список портов

COM4

115200

Подключиться

Отключиться

Вращать мотор

Против часовой

По часовой

Скорость вращения

Стоп вращения

Светодиод

ВКЛ

ВЫКЛ

Потенциометр

ВКЛ

ВЫКЛ

ОЧИСТИТЬ

ОЧИСТИТЬ

Листинги:

Скетч ардуино:

```
const int PIN_STEP = 23;
```

```
const int PIN_DIR = 22;
```

```
const int PIN_EN = 21;
```

```
const int LED = 13;
```

```
const int potens = 36;
```

```
int currentSpeed = 5;
```

```
bool clockwise = true;
```

```
bool rotating = false;
```

```
bool led_on = true;
```

```
bool is_on = false;
```

```
bool able_get_potens_val = false;
```

```
String commandBuffer = "";
```

```
void setup() {
```

```
    pinMode(PIN_STEP, OUTPUT);
```

```
    pinMode(PIN_DIR, OUTPUT);
```

```
    pinMode(PIN_EN, OUTPUT);
```

```
    pinMode(LED, OUTPUT);
```

```
    pinMode(potens, INPUT);
```

```
    digitalWrite(PIN_STEP, 1);
```

```
    digitalWrite(PIN_DIR, 0);
```

```
    Serial.begin(115200);
```

```
    delay(1000);
```

```
}
```

```
void executeCommand(String command) {
```

```
    command.trim(); // Удалить начальные и конечные пробелы
```

```
    if (!command.isEmpty()) { // Проверка на непустую строку
```

```
        if (command.startsWith("s")) {
```

```
            int newSpeed = command.substring(1).toInt();
```

```
            currentSpeed = newSpeed;
```

```
        } else if (command.startsWith("mode")) {
```

```
int mode = command.substring(4).toInt();
if (mode == 0) {
    rotating = false; // Сброс флага вращения
    is_on = false;
} else if (mode == 10) {
    rotating = true;
    clockwise = false;
    is_on = true;
} else if (mode == 11) {
    rotating = true;
    clockwise = true;
    is_on = true;
}
} else if (command.startsWith("led")) {
    int isledon = command.substring(3).toInt();
    if (isledon == 0)
        led_on = false;
    else
        led_on = true;
} else if (command.startsWith("p")) {
    int ispon = command.substring(1).toInt();
    if (ispon == 0)
        able_get_potens_val = false;
    else
        able_get_potens_val = true;
}
```

```
// // Выводим команду в монитор порта
// Serial.print("Get from serial: ");
```

```
// Serial.println(command);  
  
}  
  
}
```

```
void loop() {  
  while (Serial.available() > 0) {  
    char incomingChar = Serial.read();  
    if (incomingChar == '#') {  
      executeCommand(commandBuffer);  
      commandBuffer = ""; // Очистить буфер после выполнения команды  
    } else {  
      commandBuffer += incomingChar;  
    }  
  }  
}
```

```
digitalWrite(PIN_EN, is_on ? LOW : HIGH);
```

```
if (rotating) {  
  digitalWrite(PIN_DIR, clockwise ? HIGH : LOW);  
  digitalWrite(PIN_STEP, HIGH);  
  delay(currentSpeed);  
  digitalWrite(PIN_STEP, LOW);  
  delay(currentSpeed);  
}
```

```
digitalWrite(LED, led_on ? LOW : HIGH);
```

```
if (able_get_potens_val) {  
  int analogValue = analogRead(potens);
```

```
    Serial.print("p" + String(analogValue) + "#");  
  }  
}
```

Скрипт main.py

```
import sys
import os

from PyQt5 import uic
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtCore import Qt, QThread, pyqtSignal

from datetime import datetime

import serial.tools.list_ports
import serial

current_file_path = os.path.abspath(__file__)
folder_path = os.path.dirname(current_file_path)

class SerialThread(QThread):
    data_received = pyqtSignal(str)

    def __init__(self, port, baudrate):
        super(SerialThread, self).__init__()
        self.port = port
        self.baudrate = baudrate
        self.serial_port = None

    def run(self):
        try:
            self.serial_port = serial.Serial(self.port, self.baudrate, timeout=1)
            self.serial_port.flushInput()
            self.serial_port.flushOutput()
        except Exception as e:
            self.data_received.emit(f"Ошибка при подключении к COM-порту: {str(e)}")
            return

        current_message = ""

        while self.serial_port.is_open:
            data = self.serial_port.read(1).decode('utf-8', errors='ignore')

            if data:
                current_message += data

                if current_message.startswith('p'):

                    if(current_message.endswith("#")):
                        current_message = current_message[1:-1]

                        potentiometer_value = int(current_message)
```

```

        self.data_received.emit(f"Значение потенциометра:
{potentiometer_value}")
        current_message = ""
        self.serial_port.flushInput()
    else:
        current_message = ""
        self.serial_port.flushInput()

def send_message(self, message):
    if self.serial_port is not None and self.serial_port.is_open:
        try:
            self.serial_port.write(message.encode('utf-8'))
        except Exception as e:
            self.data_received.emit(f"Ошибка при отправке данных: {str(e)}")

def stop(self):
    if self.serial_port is not None:
        self.serial_port.close()
    self.terminate()

class MainWindowLocal(QWidget):
    def __init__(self):
        super(MainWindowLocal, self).__init__()
        uic.loadUi(folder_path + "/mainform.ui", self)

        self.serial_thread = None

        #region Start settings
        self.refreshSerialPorts()

        baudrates = ["9600", "19200", "38400", "57600", "115200"]
        self.cbBaudrate.addItem(baudrates)
        self.cbBaudrate.setCurrentIndex(len(baudrates) - 1)
        #endregion

```



```
#region Events
self.btRefreshSerialPorts.clicked.connect(self.refreshSerialPorts)
self.btClearLogs1.clicked.connect(self.clearLog1)
self.btClearLogs2.clicked.connect(self.clearLog2)

self.btConnect.clicked.connect(self.connect_serial)
self.btDisconnect.clicked.connect(self.disconnect_serial)

self.btLedOn.clicked.connect(self.sendLedOn)
self.btLedOff.clicked.connect(self.sendLedOff)

self.btPotensOn.clicked.connect(self.sendPotensOn)
self.btPotensOff.clicked.connect(self.sendPotensOff)

self.btTurnMotorLeft.clicked.connect(self.TurnMotorLeft)
self.btTurnMotorRight.clicked.connect(self.TurnMotorRight)
self.btStopTurningMotor.clicked.connect(self.StopMotorTurning)

self.sldSpeedMotor.valueChanged.connect(self.SliderSpeedChange_Handler)
#endregion
```

```
def refreshSerialPorts(self):
    self.cbSerial.clear()

    ports = serial.tools.list_ports.comports()
    for port in ports:
        self.cbSerial.addItem(port.device)
```

```
def clearLog1(self):
    self.tbLogs1.clear()
def clearLog2(self):
    self.tbLogs2.clear()
```

```
def writeToLog1(self, message):
    self.tbLogs1.append(message)
def writeToLog2(self, message):
    self.tbLogs2.append(message)
```

```

def connect_serial(self):
    if self.serial_thread is not None and self.serial_thread.isRunning():
        self.writeToLog1("Уже подключено к COM-порту.")
        return

    port_name = self.cbSerial.currentText()
    baudrate = int(self.cbBaudrate.currentText())

    self.serial_thread = SerialThread(port_name, baudrate)
    self.serial_thread.data_received.connect(self.serial_data_receive_handler
)

    self.serial_thread.start()
    self.writeToLog1(f"Подключено к {port_name} с baudрейтом {baudrate}.")

def disconnect_serial(self):
    if self.serial_thread is not None and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.writeToLog1("Отключено от COM-порта.")
        self.serial_thread = None

def send_message_to_serial(self, mess):
    if self.serial_thread is not None and self.serial_thread.isRunning():
        self.serial_thread.send_message(mess)
        self.writeToLog1(f"Отправлено: {mess}")

def serial_data_receive_handler(self, message):
    self.writeToLog2(message)

def sendLedOn(self):
    self.send_message_to_serial("led1#")
def sendLedOff(self):
    self.send_message_to_serial("led0#")

def sendPotensOn(self):
    self.send_message_to_serial("p1#")
def sendPotensOff(self):
    self.send_message_to_serial("p0#")

def TurnMotorLeft(self):
    self.send_message_to_serial("mode10#")
def TurnMotorRight(self):
    self.send_message_to_serial("mode11#")
def StopMotorTurning(self):
    self.send_message_to_serial("mode0#")

def SliderSpeedChange_Handler(self):
    value = self.sldSpeedMotor.value()
    self.send_message_to_serial(f"s{value}#")

def main():

```

```
app = QApplication(sys.argv)
mainWindow1 = MainWindowLocal()
# mainWindow1.showFullScreen()
mainWindow1.show()
sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```