Proceso de transmisión y recepecion

1. Se crean las variables en Arduino

```
float variable1 = 20.0;
float variable2 = 10.0;
float deltaChangue = 2.0;


 uint8_t arr[4] = {0};
 static bool variable1Habilitada = true;
 static bool variable2Habilitada = true;
 static bool temperaturaHabilitada = true;
 float variable1HabilitadaNumero;
 float variable2HabilitadaNumero;
float temperaturaHabilitadaNumero;
 float temp = 0.0;
```

2. Verificamos si hay datos en el puerto, luego comprobamos si el dato corresponde a la tecla 1; si es así, enviamos todas las variables al buffer

mediante un arreglo.

```
char TeclaRecibida;

    if (Serial.available() > 0)
    {
        TeclaRecibida = Serial.read();

       if (TeclaRecibida == '1')
        {
            memcpy(arr,(uint8_t *)&variable1,4);
            for(int8_t i = 0; i < 4; i++)
            {
              Serial.write(arr[i]);
            }

            memcpy(arr,(uint8_t *)&variable2,4);
            for(int8_t i = 0; i < 4; i++)
              Serial.write(arr[i]);


            memcpy(arr,(uint8_t *)&temp,4);
            for(int8_t i = 0; i < 4; i++)
            {
              Serial.write(arr[i]);
            }
```

3. En Unity creamos variables locales para almacenar los valores traídos de Arduino

```
public float habilitadovariable1;
public float habilitadovariable2;
public float habilitadotemperatura;
public float deltaChangue;
```

4. En void update verificamos si se presiono la tecla S, si es así, enviamos el valor 0x31 que representa al numero 1 que es lo que desencadena el envió de datos desde Arduino, creamos un arreglo de bytes de 28 posiciones

llamado buffer donde almacenaremos todos los datos que recibimos EN ORDEN

```
if (Input.GetKeyDown(KeyCode.S))
{
    byte[] data = { 0x31 };
    _serialPort.Write(data, 0, 1);
    byte[] buffer = new byte[28];
    Debug.Log("S");
    if (_serialPort.BytesToRead >= 4)
    {
        _serialPort.Read(buffer, 0, 28);

    variable1Texto.SetText("");
    variable2Texto.SetText("");
    temperaturaTexto.SetText("");

   float variable1 = BitConverter.ToSingle(buffer, 0);
    float variable2 = BitConverter.ToSingle(buffer, 4);
    float temperatura = BitConverter.ToSingle(buffer, 8);

    variable1Texto.text = variable1.ToString();
    variable2Texto.text = variable2.ToString();
    temperaturaTexto.text = temperatura.ToString();
```

Lo convertimos a Single para ver su representación flotante y posteriormente los convertimos a string para visualizarlos en texto, además vaciamos las variables de texto cada vez que se presiona S para asegurarnos que los datos no se acumulen

Manejo interfaz grafica

1. Cargamos las bibliotecas de TextMeshPro

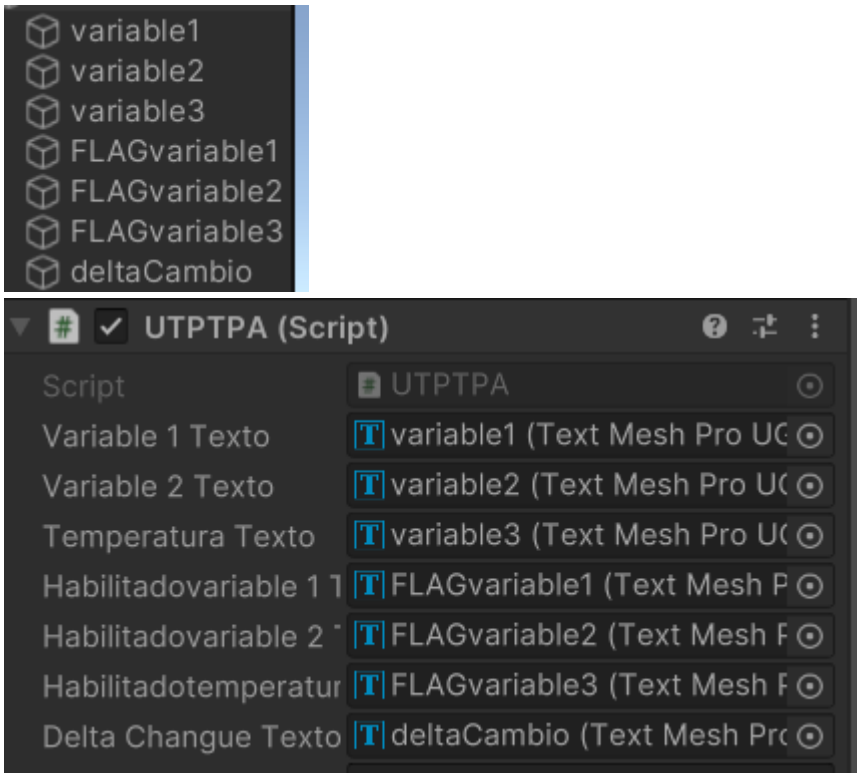
```
3   using TMPro;
4   using UnityEngine.UI;
```

2. Creamos variables de texto para todas las variables, en estas variables almacenaremos los valores recibidos convertidos a String

```
public TextMeshProUGUI variable1Texto;
public TextMeshProUGUI variable2Texto;
public TextMeshProUGUI temperaturaTexto;
```

3.
```
variable1Texto.text = variable1.ToString();
variable2Texto.text = variable2.ToString();
temperaturaTexto.text = temperatura.ToString();
```

4. En unity creamos objetos de texto y posteriormente los asignamos en el inspector a las variables de texto



Construcción y funcionamiento del micro

1. En Arduino establecemos la velocidad de ejecución, además establecemos la lectura de el pin4 encargado de la temperatura

```
void setup()
{
  Serial.begin(115200);
    adc_init();
  adc_set_temp_sensor_enabled(true);

}
```

2. En Unity creamos un objeto _SerialPort de la clase serialPort mediante un método constructor

```
private SerialPort _serialPort = new SerialPort();
```

3. En void start establecemos los atributos de _SerialPort, es importante que la velocidad establecida en Arduino sea la misma velocidad establecida en unity , además estableceremos otros valores, como el nombre del puerto, y si este está activado

```
void Start()
{
    _serialPort.PortName = "COM4";
    _serialPort.BaudRate = 115200;

    _serialPort.DtrEnable = true;
    _serialPort.Open();
    Debug.Log("Puerto serial listo");
}
```

4. Se usa la función OnDestroy para asegurarse que la comunicación entre Arduino y Unity se corte si el objeto al que esta adjutando el script es eliminado, evitando errores y congelamientos

```
}
    void OnDestroy()
{
    // Cierra el puerto serial cuando el objeto se destruye
    if (_serialPort != null && _serialPort.IsOpen)
    {
        _serialPort.Close();
        Debug.Log("Puerto serial cerrado.");
    }
}
```

-