Introducción:

En esta actividad, nos embarcaremos en el desarrollo de un programa basado en un microcontrolador Arduino o una Raspberry Pi Pico. Utilizaremos dos códigos, uno en el entorno de Arduino y otro en Unity, para establecer una comunicación efectiva entre ambos sistemas.

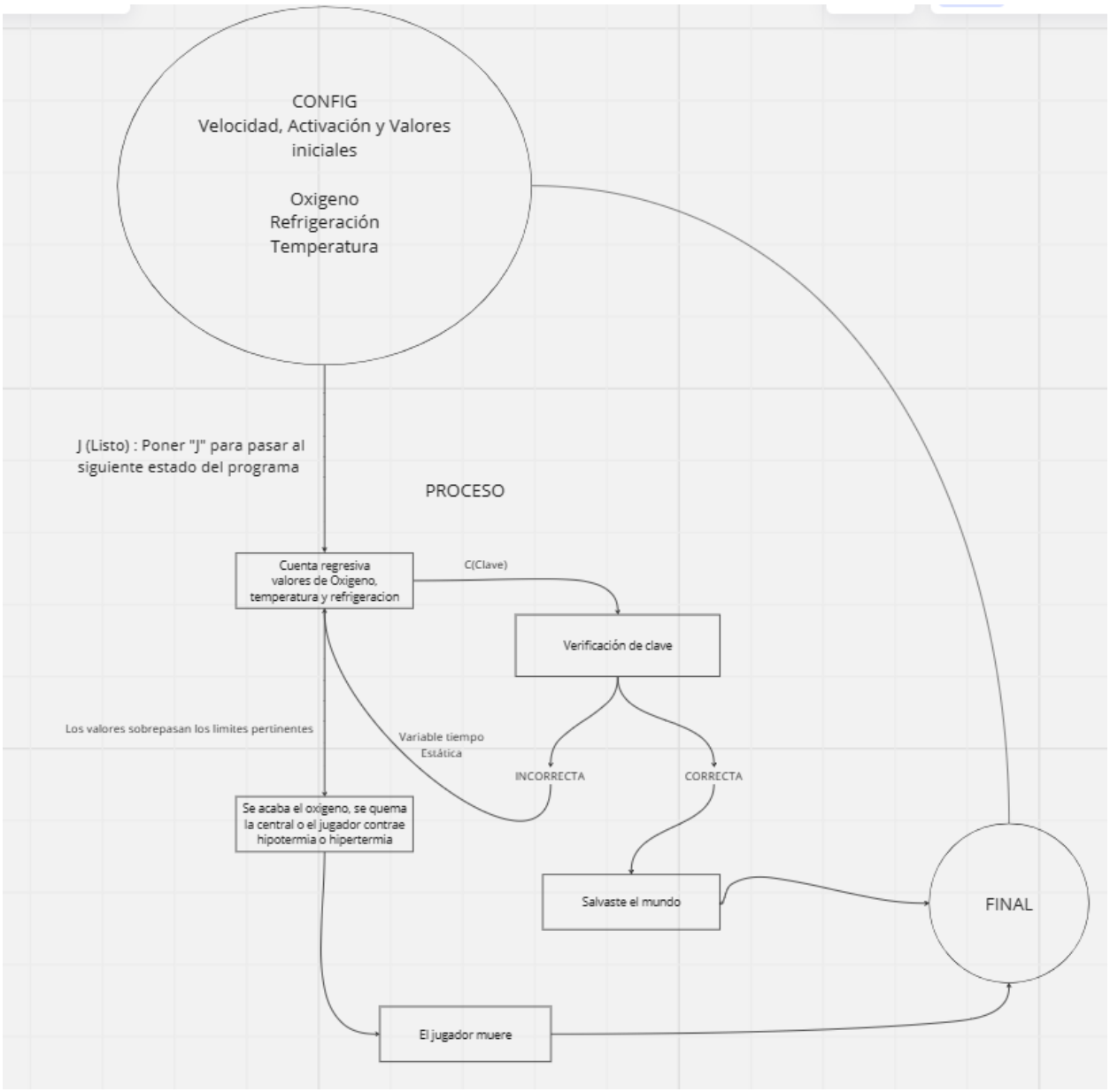
Objetivo:

El propósito de este proyecto es crear un programa que actúe como puente de comunicación entre una placa Raspberry Pi Pico y dos códigos, uno ejecutado en el entorno de Arduino y otro en Unity. La meta principal es establecer una conexión unilateral que permita escribir, enviar, recibir y leer datos desde ambas partes de manera eficiente y coherente

- 3 Variables y la posibilidad de cambiar su valor inicial, su velocidad de cambio y habilitar o deshabilitar la variable
- Una interfaz grafica que permita interactuar con estos datos
- Narrativa que acompañe la experiencia

Funcionamiento del Programa

Diagrama de estados:



Codigo de arduino:

Declaracion de Variables:

```
static uint32_t Temperatura;
static uint32_t VelocidadConsumoTemperatura = 1;
static uint32_t Refrigeracion = 1;
static uint32_t VelocidadConsumoRefrigeracion = 1;
static uint32_t Oxigeno = 1;
static uint32_t VelocidadConsumoOxigeno = 1;
static bool oxigenoHabilitado = true;
static bool refrigeracionHabilitada = true;
static bool temperaturaHabilitada = true;
```

Para todas las variables, se declaran variables para manipular su valor inicial, su temperatura y una variable de tipo booleano que permita su activación y desactivación

```
case TaskIStates::CONFIG:
{

  char TeclaRecibida;

    if (Serial.available() > 0)
    {

        TeclaRecibida = Serial.read();

                if (TeclaRecibida == 'O')
    {
        oxigenoHabilitado = !oxigenoHabilitado; // Cambiar el estado de habilitación
        Serial.print("Oxigeno habilitado: ");
        Serial.println(oxigenoHabilitado ? "Si" : "No");
    }
```

Mediante una máquina de estados en el estado de configuración, se permite habilitar y deshabilitar la variable, así como cambiar su valor inicial y su velocidad utilizando la entrada de teclado.

```
// ----------------------------INICIO MANEJO OXIGENO-------------------

 if (oxigenoHabilitado)
   {
    if (TeclaRecibida == 'D')
    {
        VelocidadConsumoOxigeno++;
        Serial.println(VelocidadConsumoOxigeno);

    }
```

```
if (TeclaRecibida == 'J')
{
Serial.println("HOLA, PASANDO AL JUEGO");
task1State = Task1States::OxigenoProceso;
}
```

Al presionar la tecla J, una vez que toda la configuración haya sido establecida, se avanza al siguiente estado de la máquina llamado 'OxigenoProceso', donde se llevará a cabo toda la lógica para disminuir las variables.

```
  case Task1States::OxigenoProceso:
{
    static unsigned long lastUpdateTime = 0; // Variable para almacenar el tiempo del último update
    static unsigned long updateInterval = 1000; // Intervalo de actualización en milisegundos (1 segundo)

    // Bucle mientras haya oxígeno, temperatura y refrigeración
    while (Oxigeno > 0 || Temperatura > 0 || Refrigeracion > 0)
    {
        unsigned long currentTime = millis(); // Obtener el tiempo actual

        // Verificar si ha pasado el tiempo suficiente para actualizar
        if (currentTime - lastUpdateTime >= updateInterval)
        {
            lastUpdateTime = currentTime; // Actualizar el tiempo del último update

            // Reducir oxígeno, temperatura y refrigeración según la velocidad de consumo
            if (Oxigeno > 0) {
                Oxigeno -= VelocidadConsumoOxigeno;
            }
            if (Refrigeracion > 0) {
                Refrigeracion -= VelocidadConsumoRefrigeracion;
            }
            if (Temperatura > 0) {
                Temperatura -= VelocidadConsumoTemperatura;
            }

            // Enviar datos a Unity
            Serial.print("B");
            Serial.println(Oxigeno);
            Serial.print("N");
            Serial.println(Refrigeracion);
            Serial.print("M");
            Serial.println(Temperatura);
        }
    }
```

Mediante el uso de variables temporales establecidas por millis (que cumplen la función de una espera de 1000 milisegundos sin tener que recurrir a la función delay, ya que podría causar bloqueos en el programa), se disminuye el valor de oxígeno/refrigeración/temperatura en el valor escogido en velocidad de consumo. Posteriormente, se imprimen/almacenan en el buffer, precedidos por una etiqueta distintiva como "B", "M" o "N", que servirá para identificar la variable desde el código de Unity

```
// Comprobar si se agotó alguna de las variables
if (Oxigeno <= 0 && Temperatura <= 0 && Refrigeracion <= 0)
{
    Serial.println("Se acabó el oxígeno, temperatura o refrigeración, MORISTE");
    task1State = Task1States::FINAL;
}
```

Si las 3 variables llegan a cero, entonces la máquina de estados pasa a su estado final, donde todos los valores se restablecen.

```
    case Task1States::FINAL:
 {



    TiempoParaAbrir = 120;
    lastTime = 0;
    lastTimeOxigeno  = 0;
    Temperatura = 0;
    Refrigeracion = 0;
    Oxigeno = 0;
    VelocidadConsumoOxigeno = 0;

    VelocidadConsumoTemperatura = 1;

     VelocidadConsumoRefrigeracion = 1;

      oxigenoHabilitado = true;
    refrigeracionHabilitada = true;
    temperaturaHabilitada = true;

    task1State = Task1States::CONFIG;

    break;
 }
```

CODIGO DE UNITY

Una vez que el código de Arduino esté programado, solo es necesario crear un programa en Unity para enviar datos al microcontrolador, recibir nuevos datos y convertirlos a

valores numéricos. Esto se debe a que lo que se almacena en el buffer son impresiones de cadenas de texto, por lo que es necesario convertirlas para poder manipularlas en Unity.

```
private int oxygenConsumptionRate;
private int oxygenRate;

private int oxygenRateInGame;
private int RefrigerationRateInGame;
private int TemperatureRateInGame;


private int RefrigerationConsumptionRate;
private int RefrigerationRate;

private int TemperatureConsumptionRate;
private int TemperatureRate;
```

En esta parte del código, declaramos variables de tipo entero que serán las encargadas de almacenar, en Unity, los valores traídos desde la placa de Arduino.

```
public void DisminuirOxigeno()
{
        byte[] data = { 0x48};
        _serialPort.Write(data, 0, 1);
        Debug.Log("Send Data");

        if (_serialPort.BytesToRead > 0)
        {
            int bytesAvailable = _serialPort.BytesToRead;
            int numData = _serialPort.Read(buffer, 0, bytesAvailable);
            string receivedData = System.Text.Encoding.ASCII.GetString(buffer,
            Debug.Log("Received Data: " + receivedData);

            if (int.TryParse(receivedData, out oxygenRate))
            {

                Debug.Log("Oxygen  Rate: " + oxygenRate);
            }
            else
            {
                Debug.Log("Failed to parse received data as integer.");
            }


            Debug.Log("Bytes received: " + numData.ToString());
        }
        else
        {
            Debug.Log("No data available to read.");
        }

        Oxigeno.text = oxygenRate.ToString();
}
```

Se envía mediante un arreglo de bytes el valor hexadecimal 0x48, que representa la letra que desencadena el evento en Arduino. Este valor hexadecimal se envía a la placa mediante la instrucción Serial.Write. Posteriormente, se recupera lo almacenado en el buffer utilizando Serial.Read. Sin embargo, para un manejo más fácil de lo recibido, se reenvía a una variable de tipo entero. Esta variable podrá ser manipulada en Unity. (Obsérvese la instrucción 'Out OxygenRate', donde 'OxygenRate' representa la variable de tipo entero hacia la que se redirige lo recibido en el buffer). Además, de este modo, el buffer podrá estar vacío cuando necesite recibir otra instrucción

```
public void IniciarPartida()
{



        byte[] data = { 0x4A};
        _serialPort.Write(data, 0, 1);
        Debug.Log("Send Data");



        if (_serialPort.BytesToRead > 0)
        {

            int bytesAvailable = _serialPort.BytesToRead;


            int numData = _serialPort.Read(buffer, 0, bytesAvailable);


            string receivedData = System.Text.Encoding.ASCII.GetString(buffer, 0, numData);
            Debug.Log("Received Data: " + receivedData);

        }
        else
        {
            Debug.Log("No data available to read.");
        }
}
```

Función en Unity que envía, mediante hexadecimales, la letra 'J', la cual en el código de Arduino ejecuta el cambio de estado de la máquina a 'OxigenoProceso' o 'Jugar'.

```
foreach (string data in dataSplit)
{
    if (data.Length > 0)
    {
        // Identifica el tipo de dato por el primer carácter
        char dataType = data[0];
        // Obtiene el valor del dato eliminando el primer carácter
        if (int.TryParse(data.Substring(1), out int value))
        {
            // Actualiza el valor correspondiente según el tipo de dato
            switch (dataType)
            {
                case 'B':
                    oxygenRateInGame = value;
                    Debug.Log("Oxygen Rate: " + oxygenRateInGame);
                    break;

                case 'N':
                    RefrigerationRateInGame = value;
                    Debug.Log("Refrigeration Rate: " + RefrigerationRateInGame);
                    break;

                case 'M':
                    TemperatureRateInGame = value;
                    Debug.Log("Temperature Rate: " + TemperatureRateInGame);
                    break;

                default:
                    Debug.Log("Invalid data type: " + dataType);
                    break;
            }
```

Dentro de la función Update, se reciben los valores enviados desde Arduino en este segmento:

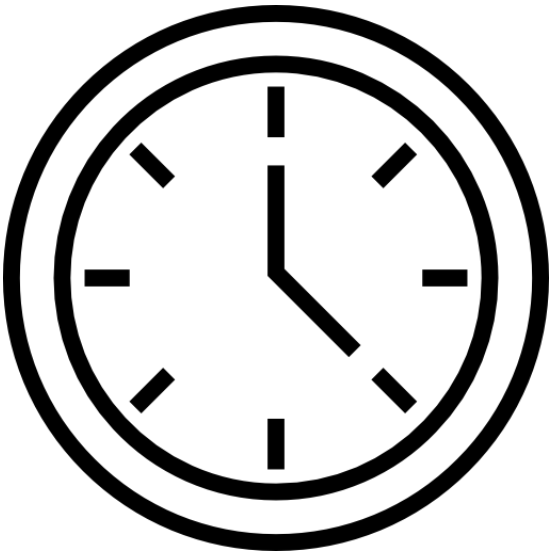
```
        // Enviar datos a Unity
        Serial.print("B");
        Serial.println(Oxigeno);
        Serial.print("N");
        Serial.println(Refrigeracion);
        Serial.print("M");
        Serial.println(Temperatura);
```

Como se mencionó anteriormente, las letras antes de cada valor servirán para identificar a qué variable se deben dirigir los datos del búfer. Esto se realiza mediante casos, y a través de un arreglo se separa el primer carácter recibido (la letra/etiqueta) del resto de caracteres (los valores numéricos de cada variable).

Posteriormente en Unity se revisa si el jugador introdujo una contraseña en un elemento de UI de tipo InputField, si es incorrecto no pasara nada, si es correcto entonces el jugador ganara.

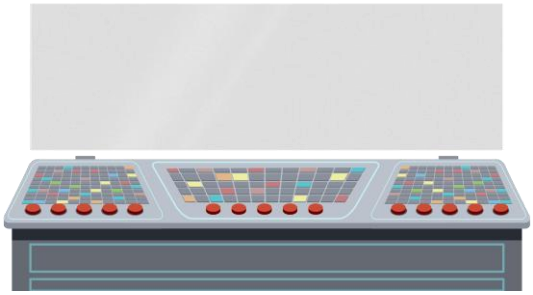
```
// Método que se llama cuando se presiona un botón para verificar el texto del InputField
public void VerificarTexto()
{
    // Accede al texto del InputField usando su propiedad text
    string textoIngresado = inputField.text;

    // Comprueba si el texto ingresado es igual a "C1234"
    if (textoIngresado == "C1234")
    {
        Debug.Log("Correcto");
        PantallaDeVictoria.SetActive(true);
    }
    else
    {
        Debug.Log("Incorrecto");
    }
}
```

Construccion de la interfaz grafica

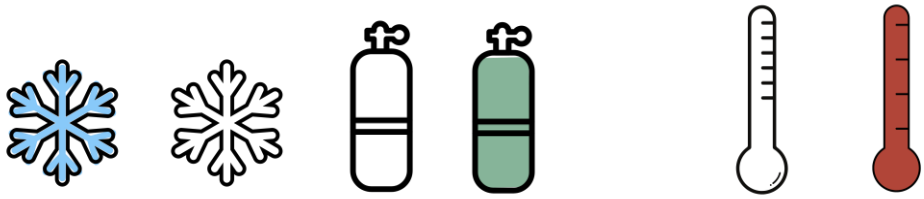


Para representar la velocidad en la que las variables deben cambiar

Para aumentar o disminuir distintas variables



Como objeto principal en la interfaz, aunque sin funcionalidad aporta a la narrativa y ambientacion de una central nuclear



Objeto Grafico para Representar la refrigeracion, el oxigeno y la temperatura.