Haoyan Chen

PUZZLE 2

1.CONFIGURACIÓN

Para configurar el puzzle 2 tendremos que instalar una librería que permite crear interfaces gráficas, para poder visualizar ventanas de colores diferentes para detectar la nfc.
Hemos optado por usar la librería “gtk3”. Debemos instalarlo como una gema para poder reutilizar las clases y las funciones del puzzle 2

Instalación de Bundler (si no está instalado):
Ejecuta el siguiente comando para instalar Bundler, el gestor de dependencias para

```
gem install bundler
```

Creación de la estructura de la gema:
Utiliza el siguiente comando para generar la estructura base de la gema llamada "puzzle1":

```
bundle gem puzzle1
```

Incorporación del código:
Agrega el código correspondiente al Puzzle 1 dentro del archivo:

```
lib/puzzle1.rb
```

Construcción e instalación de la gema localmente:
Primero, construye la gema utilizando el siguiente comando, el cual empaqueta la gema para su instalación:

```
rake build
```

Luego, instala la gema localmente con el siguiente comando (verifica que la versión y el nombre del archivo sean correctos):

```
gem install pkg/puzzle1-0.1.0.gem
```

Integración en Puzzle 2:
Para incorporar la funcionalidad de Puzzle 1 en Puzzle 2, añade la siguiente línea en el archivo correspondiente:

```
require 'puzzle1'
```

PROBLEMAS ENCONTRADOS

Uno de los problemas más grandes que tuve haciendo el puzzle2 es que estaba intentando descubrir qué le había pasado a mi lector NFC, ya que no detectaba el I2C. Pero luego cambiamos de lector y todo ya fue bien.

En el apartado anterior es una de las maneras para configurar el puzzle2 .Para que pueda utilizar las funciones del puzzle1 mediante el require 'puzzle1'. Pero cuando lo hice tenía muchos problemas con la implementación de la gema, así que optamos por otra opción porque ya había pasado mucho rato intentando solucionar el problema.

En nuestro caso utilizamos otro método donde añadiremos una línea de código en el puzzle1 para que sea un módulo "una librería accesible"

"Module puzzle1"

```
require 'ruby-nfc'

module Puzzle1
  class Rfid
    def read_uid
      readers = NFC::Reader.all
      return nil if readers.empty?

      readers[0].poll(IsoDep::Tag, Mifare::Classic::Tag,
Mifare::Ultralight::Tag) do |tag|
        begin
          case tag
          when Mifare::Classic::Tag, Mifare::Ultralight::Tag
            return tag.uid_hex.upcase
          when IsoDep::Tag
            return tag.uid.unpack1('H*').upcase
          end
        rescue StandardError => e
          puts "Error al llegir la targeta: #{e.message}"
          return nil
        end
      end
      nil
    end
  end
end
if __FILE__ == $0
  rf = Puzzle1::Rfid.new
  puts "Esperando para leer la tarjeta NFC..."
  uid = rf.read_uid
```

```
  if uid
    puts "UID de la tarjeta: #{uid}"
  else
    puts "No se pudo leer la tarjeta."
  end
end
```

IMPLEMENTACIÓN DEL CÓDIGO :

Ahora crearemos el puzzle2 donde utilizaremos el puzzle1:

```
nano ~/Pbe/puzzle2.rb
```

Un ejemplo para el código del puzzle2 para saber si esta cogiendo la funcion del puzzle1 podría ser el siguiente :

```
require_relative "puzzle1"

puts "Executant puzzle2..."
rf = Puzzle1::Rfid.new  # Aquí utilitzem la classe Rfid de puzzle1
puts "Esperant per llegir una targeta NFC..."
uid = rf.read_uid

if uid
  puts "UID de la targeta: #{uid}"
else
  puts "No s'ha pogut llegir la targeta."
end
```

En este ejemplo nos podemos fijar en varias cosas :

```
require_relative "puzzle1"
```

Pusimos relative porque puzzle1 y puzzle2 estan en una misma carpeta, de esta manera podremos utilizar puzzle1, sin relative no funcionaria.

```
rf = Puzzle1::Rfid.new  # Aquí utilitzem la classe Rfid de puzzle1
```

El código anterior también es muy importante, normalmente con un rf=Rfid:new ya nos serviría, pero en este caso hay que utilizar el Rfid del puzzle1.
Y lo otro ya es leer la tarjeta nfc con el puzzle1 y para imprimir textos.

```
haoyan@haorsb:~/Pbe $ ruby puzzle2.rb
Executant puzzle 2....
Esperant per llegir una targeta NFC...
UID de la targeta: 9A6AC901
```

y vemos que nos ejecuta el puzzle2 perfectamente y el require 'puzzle1' también nos funciona, ahora tendremos que configurar su interfaz para visualizar los botones.

Antes de ejecutar el código de ejemplo con GTK3, necesitas instalar las librerías necesarias y la gema **gtk3** en tu sistema.

```
sudo apt-get update
sudo apt-get install libgtk-3-dev
```

Esto instala los archivos de desarrollo de **GTK 3** que son necesarios para compilar extensiones en Ruby
ahora instalamos la gema gtk3

```
gem install gtk3
```

Ahora Ruby podrá usar la librería GTK3 para construir interfaces gráficas.

CÓDIGO PARA INTERFACES GRÁFICAS

```
require "gtk3"
require "thread"
require_relative "puzzle1"

@rf = Puzzle1::Rfid.new
@window = Gtk::Window.new("Rfid Window")
@window.set_size_request(600, 150)
@window.set_border_width(5)
@uid = ""

@blue = Gdk::RGBA.new(0, 0, 1, 1)
@white = Gdk::RGBA.new(1, 1, 1, 1)
@red = Gdk::RGBA.new(1, 0, 0, 1)

@window_button = Gtk::Button.new(label: "Por favor, acerca tu tarjeta de
identidad de la uni")
@window_button.override_background_color(:normal, @blue)
@window_button.override_color(:normal, @white)
@button = Gtk::Button.new(label: "Clear")

@fixed = Gtk::Fixed.new
```

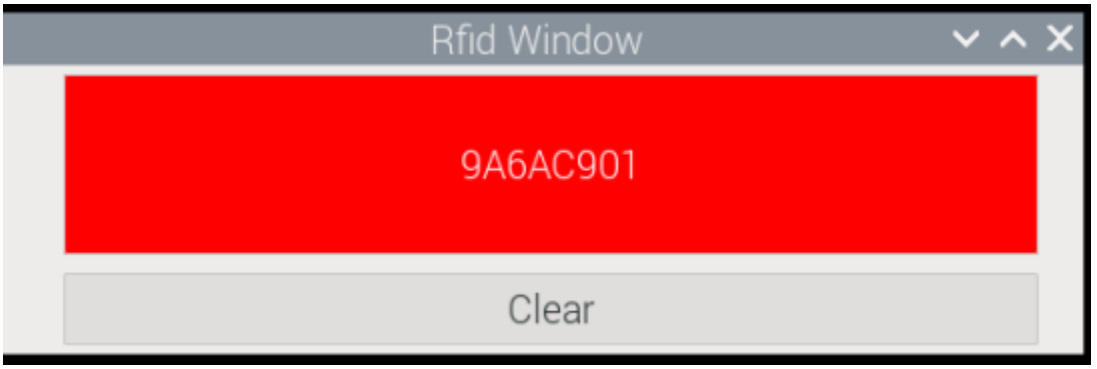
```
@button.set_size_request(540, 40)
@window_button.set_size_request(540, 100)
@fixed.put(@window_button, 30, 0)
@fixed.put(@button, 30, 110)
@window.add(@fixed)

@button.signal_connect("clicked") do
  if @uid != ""
    @uid = ""
    @window_button.override_background_color(:normal, @blue)
    @window_button.set_label("Por favor, acerca tu tarjeta de identidad
de la uni")
    threads
  end
end

def threads
  Thread.new do
    lectura
    puts "END THREAD"
    Thread.exit
  end
end

threads

def lectura
  puts "Esperando id"
  @uid = @rf.read_uid
  GLib::Idle.add { gestion_UI }
end

def gestion_UI
  if @uid != ""
    @window_button.set_label(@uid)
    @window_button.override_background_color(:normal, @red)
  end
end

@window.signal_connect("delete-event") { Gtk.main_quit }
@window.show_all
Gtk.main
```

Para ejecutar el puzzle2 implementamos el siguiente código :

```
ruby puzzle2.rb
```

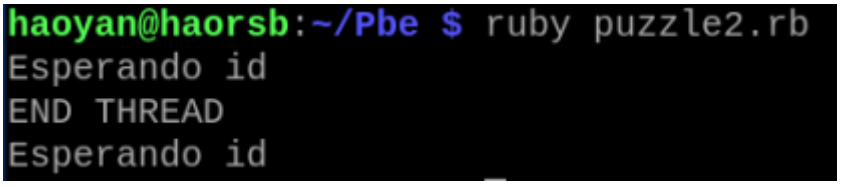
La ejecución es lo siguiente

Si introducimos la tarjeta NFC



Si podemos Clear, nos volverá a pedir la tarjeta nfc

Si pulsamos la esquina derecha X se cierra la pestaña



En esta ventana vemos que los threads se acaba uno y empieza uno, dependiendo si pulsamos por ejemplo clear .

EXPLICACIÓN CÓDIGO

Carga de bibliotecas y dependencias

```
require "gtk3"
require "thread"
require_relative "puzzle1"
```

Inicialización de variables y configuración de la ventana

```
@rf = Puzzle1::Rfid.new
@window = Gtk::Window.new("Rfid Window")
@window.set_size_request(600, 150)
@window.set_border_width(5)
@uid = ""
```

Se crea una instancia de `Puzzle1::Rfid` donde contiene el método `read_uid`.
Se crea una ventana GTK titulada "Rfid Window" y se establece su tamaño inicial (600 píxeles de ancho por 150 de alto) y un ancho de borde de 5 píxeles.
La variable `@uid` se inicializa como cadena vacía, y servirá para almacenar el UID leído.

```
@blue = Gdk::RGBA.new(0, 0, 1, 1)
@white = Gdk::RGBA.new(1, 1, 1, 1)
@red = Gdk::RGBA.new(1, 0, 0, 1)
```

Se crean tres objetos Gdk::RGBA que representan los colores azul, blanco y rojo. Estos se usarán para modificar la apariencia de los botones en la interfaz.

```
@window_button = Gtk::Button.new(label: "Por favor, acerca tu tarjeta de identidad de la uni")
@window_button.override_background_color(:normal, @blue)
@window_button.override_color(:normal, @white)
@button = Gtk::Button.new(label: "Clear")
```

@window_button: Es el botón principal que muestra el mensaje para acercar la tarjeta. Se le cambia el color de fondo a azul y el color del texto a blanco.
@button: Es un botón adicional etiquetado "Clear", que servirá para reiniciar la lectura.

```
@fixed = Gtk::Fixed.new
@button.set_size_request(540, 40)
@window_button.set_size_request(540, 100)
@fixed.put(@window_button, 30, 0)
@fixed.put(@button, 30, 110)
@window.add(@fixed)
```

Se crea un contenedor Gtk::Fixed para posicionar de forma exacta los widgets.

```
@button.signal_connect("clicked") do
  if @uid != ""
    @uid = ""
    @window_button.override_background_color(:normal, @blue)
    @window_button.set_label("Por favor, acerca tu tarjeta de identidad de la uni")
    threads
  end
```

```
end
```

Se conecta una señal al botón "Clear" para que, al hacer clic, se reinicien los valores

```
def threads
  Thread.new do
    lectura
    puts "END THREAD"
    Thread.exit
  end
end
threads
```

threads: Crea un hilo nuevo que llama a la función lectura para leer el UID. Una vez terminada la lectura, muestra "END THREAD" y finaliza el hilo.

```
def lectura
  puts "Esperando id"
  @uid = @rf.read_uid
  GLib::Idle.add { gestion_UI }
en
```

llama al método `read_uid` del objeto `@rf` para obtener el UID y permite actualizar la interfaz gráfica en el hilo principal

```
def gestion_UI
  if @uid != ""
    @window_button.set_label(@uid)
    @window_button.override_background_color(:normal, @red)
  end
end
```

gestion_UI: Se ejecuta en el hilo principal para actualizar la interfaz

```
@window.signal_connect("delete-event") { Gtk.main_quit }
@window.show_all
Gtk.main
```

Se conecta la señal de "delete-event" (por ejemplo, al cerrar la ventana) para que se cierre la aplicación correctamente mediante Gtk.main_quit.

Se llama a @window.show_all para mostrar todos los widgets de la ventana.

Finalmente, Gtk.main inicia el bucle principal de GTK, que mantiene la aplicación ejecutándose y esperando eventos.