

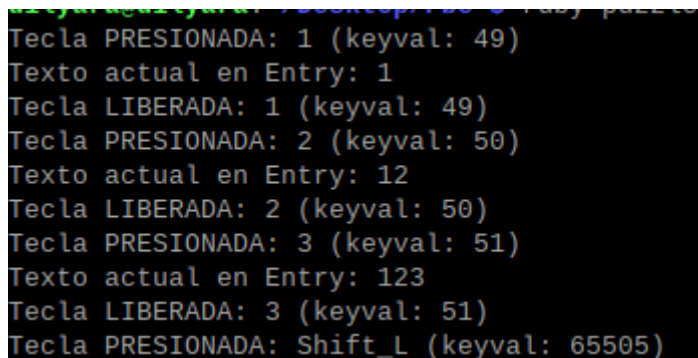
Implementación de Puzzle2 con GTK3

GTK maneja los eventos de teclado de manera diferente en comparación con la entrada estándar. No es simplemente capturar texto como si viniera de un teclado normal.

Los lectores **RFID tipo teclado USB** suelen enviar datos rápidamente, carácter por carácter, y terminan con un **Enter** (Return). Sin embargo, GTK puede procesar eventos de teclado de manera especial dependiendo de la configuración del widget y del contexto de la aplicación, puede tratar diferente los caracteres de teclado.

Problemas encontrados durante el trabajo con GTK3:

Quise ver como el GTK3 procesa los caracteres:



```
Tecla PRESIONADA: 1 (keyval: 49)
Texto actual en Entry: 1
Tecla LIBERADA: 1 (keyval: 49)
Tecla PRESIONADA: 2 (keyval: 50)
Texto actual en Entry: 12
Tecla LIBERADA: 2 (keyval: 50)
Tecla PRESIONADA: 3 (keyval: 51)
Texto actual en Entry: 123
Tecla LIBERADA: 3 (keyval: 51)
Tecla PRESIONADA: Shift_L (keyval: 65505)
```

Keyval es es el código numérico que GTK usa para representar cada tecla.

Resolución del problema:

Para evitar que GTK interprete mal los eventos del teclado (especialmente caracteres especiales o teclas modificadoras), podemos hacer lo siguiente:

1. **Usar key-press-event pero ignorando caracteres innecesarios.**
2. **Filtrar solo caracteres ASCII estándar (a-z, 0-9) y concatenarlos en un buffer temporal.**
3. **Detectar "Enter" (Return) para procesar el código RFID completo.**

El código:

```
require 'gtk3'

# cambia el color de fondo del widget usando valores RGB (0-1)
def set_background_color(widget, r, g, b)
  widget.override_background_color(:normal, Gdk::RGBA.new(r, g, b, 1))
end

# maneja el teclado
def handle_key(event)
  char = event.string
  if event.keyval == Gdk::Keyval::KEY_Return
    process_rfid(@rfid_buffer) unless @rfid_buffer.empty?
    @rfid_buffer = ""
  elsif char.match?(/[a-zA-Z0-9]/)
    @rfid_buffer += char
  end
end
```

Si la tecla presionada es "Enter" (KEY_Return)

if event.keyval == Gdk::Keyval::KEY_Return

- Significa que el usuario (o el lector RFID) presionó **Enter**.
- Si el buffer @rfid_buffer tiene datos, los envía a process_rfid(@rfid_buffer).
- Luego **limpia el buffer** (@rfid_buffer = "") para la próxima lectura

```
# reset
def reset_view
  @rfid_buffer = ""
  @label.text = "Please, login with your university card"
  set_background_color(@event_box, 0, 0, 1) # Fondo azul
end
```

```

def initialize
  # para acumular caracteres del UID
  @rfid_buffer = ""

  # crear ventana principal
  @window = Gtk::Window.new("RFID Reader")
  @window.set_default_size(400, 100)
  @window.signal_connect("destroy") { Gtk.main_quit }

  # el área de visualización y el botón
  vbox = Gtk::Box.new(:vertical, 5)
  vbox.margin = 5
  @window.add(vbox)

  # EventBox para permitir cambiar el color de fondo
  @event_box = Gtk::EventBox.new
  vbox.pack_start(@event_box, expand: true, fill: true, padding: 0)

  # label para mostrar mensajes y resultados
  @label = Gtk::Label.new("Please, login with your university card")
  @label.override_color(:normal, Gdk::RGBA.new(1, 1, 1, 1)) # Texto en blanco
  @event_box.add(@label)

  # botón "Clear" para reiniciar la visualización
  clear_button = Gtk::Button.new(label: "Clear")
  clear_button.signal_connect("clicked") { reset_view }
  vbox.pack_start(clear_button, expand: false, fill: false, padding: 0)

  # capturar eventos de teclado para leer el UID
  @window.add_events(Gdk::EventMask::KEY_PRESS_MASK)
  @window.signal_connect("key-press-event") { |_, event| handle_key(event) }

  # fondo azul
  set_background_color(@event_box, 0, 0, 1) # para cambiar el color del fondo
  @window.show_all
end

```

"key-press-event": Se ejecuta cada vez que una tecla es presionada.

|_, event|:

- **_** → Se ignora el primer parámetro (generalmente el widget que recibe el evento).
- **event** → Contiene información sobre la tecla presionada.

handle_key(event): Llama a la función que procesará la tecla ingresada.

```

# elimina ceros iniciales, convierte a decimal y hexadecimal
def process_rfid(uid)
  clean_uid = uid.gsub(/^0+/, '') # eliminamos los ceros
  begin
    dec = Integer(clean_uid) #los convertimos a decimal
  rescue ArgumentError
    @label.text = "Error: Invalid UID."
    return
  end

  hex_big = dec.to_s(16).upcase.rjust(8, '0') # Big Endian
  hex_little = hex_big.scan(/../).reverse.join # Little Endian

  @label.text = "UID Decimal: #{dec}\nHex Big: #{hex_big}\nHex Little: #{hex_little}"
  set_background_color(@event_box, 1, 0, 0) # fondo rojo
end

```

En resumen, he implementado una interfaz gráfica en Ruby con GTK3 para leer e interpretar códigos RFID. La aplicación utiliza la variable **@rfid_buffer** para acumular los caracteres recibidos a través del evento **key-press-event** de la ventana. Al pulsar la tecla “Enter”, se activa el método **process_rfid**, que limpia los ceros iniciales del UID, lo convierte a decimal y lo formatea en hexadecimal en representaciones Big Endian y Little Endian.

La interfaz se compone de:

- Una ventana principal con un contenedor vertical.
- Un **Gtk::EventBox** que permite cambiar el color de fondo (por ejemplo, azul al inicio y rojo al procesar un UID).
- Un **Gtk::Label** para mostrar mensajes y resultados.
- Un botón "Clear" que reinicia la vista y borra la información anterior.

Con este diseño se ha resuelto el problema de integración con la librería GTK3, ofreciendo una solución clara y funcional para la lectura e interpretación de UID.

