Memòria del puzzle 2

Descripció del Projecte

L'objectiu del projecte és crear una aplicació gràfica amb **Ruby GTK3** que permeti introduir text multilínia i mostrar-lo en una pantalla LCD connectada a una **Raspberry Pi** mitjançant **I2C**. El programa permet a l'usuari escriure text en un camp de text i, en fer clic a un botó, el contingut es mostra en la pantalla LCD

Instal·lació de GTK3 en Raspberry Pi 5

Per configurar l'entorn de desenvolupament en **Ruby GTK3** en una Raspberry Pi 5, vaig seguir aquests passos:

1. **Actualització de Paquets del Sistema**: Primer, vaig actualitzar el sistema amb:

```
Unset
sudo apt update && sudo apt upgrade
```

2. **Instal·lació de GTK3**: Vaig instal·lar les llibreries necessàries amb:

```
Unset
sudo apt install libgtk-3-dev
```

3. **Instal·lació de Ruby i la Llibreria GTK3 per Ruby**: Vaig instal·lar Ruby i la gemma `gtk3`:

```
Unset
sudo apt install ruby-full

gem install gtk3
```

4. **Configuració de la Llibreria FFI**: Vaig instal·lar la gemma `ffi` per assegurar compatibilitat amb `gtk3`:

```
Unset
gem install ffi
```

5. **Prova d'Instal·lació**: Finalment, vaig comprovar que tot estava configurat correctament executant un petit script de prova que mostrava una finestra amb un missatge, confirmant així la correcta instal·lació de GTK3 amb Ruby.

Implementació del Codi

1. **Configuració de la Pantalla LCD (I2C)**:

Utilitzo la llibreria I2C::Drivers::LCD::Display per establir la connexió amb la pantalla LCD. La pantalla s'inicialitza amb els paràmetres de ruta (/dev/i2c-1), l'adreça (0x27), el nombre de files (4) i el nombre de columnes (20):

```javascript
lcd = I2C::Drivers::LCD::Display.new('/dev/i2c-1', 0x27, rows=4, cols=20)
```

2. **Funció `lcd_print` per Visualitzar Text en la Pantalla LCD**:

Definim la funció `lcd_print` per netejar la pantalla (`lcd.clear`) i imprimir el text en línies separades. Aquesta funció s'encarrega de dividir el text en línies i imprimir cada línia en la seva corresponent fila a la pantalla LCD (fins a un màxim de 20 caràcters per línia):

```javascript
def lcd_print(lcd, text)
  lcd.clear
  l = text.split("\n")
  for i in 0...l.length
    lcd.text(l[i][0..19], i)
  end
end
```

3. **Creació de la Finestra Principal**:

La finestra es crea amb `Gtk::Window.new`, i s'especifica un títol i un marge (`set_border_width(10)`) per mantenir la interfície organitzada i agradable visualment.

4. **Configuració del `TextView` per a Entrada de Text Multilínia**:

He afegit un `TextView` amb habilitats d'edició per permetre l'entrada de text per part de l'usuari. S'activa el mode de paraula (`set_wrap_mode(:word)`) per ajustar el text automàticament si supera l'ample del camp de text:

```
JavaScript
textview = Gtk::TextView.new
textview.set_wrap_mode(:word)
textview.set_editable(true)
textview.set_cursor_visible(true)
```

5. **Estilització del `TextView` amb CSS**:

He aplicat una font de tipus monoespai al `TextView` utilitzant CSS. Es defineix el CSS amb `Gtk::CssProvider`, i s'aplica amb `style_context.add_provider`:

```
JavaScript
css = <<-CSS
  textview {
    font-family: "monospace";
    font-size: 12px;
  }
CSS
provider = Gtk::CssProvider.new
provider.load(data: css)
textview.style_context.add_provider(provider,
Gtk::StyleProvider::PRIORITY_USER)
```

6. **Afegir un `ScrolledWindow` pel `TextView`**:

Per evitar problemes de visualització amb grans quantitats de text, he col·locat el `TextView` dins d'un `ScrolledWindow`. Això activa automàticament la barra de desplaçament quan el text excedeix l'espai disponible:

```
JavaScript
scrolled_window = Gtk::ScrolledWindow.new
scrolled_window.set_policy(:automatic, :automatic)
scrolled_window.add(textview)
```

7. **Botó d'Acció ("Mostra el text")**:

He afegit un botó amb l'etiqueta "Mostra el text". Quan es fa clic, aquest botó recupera el text introduït en el `TextView` i el mostra a la pantalla LCD. També imprimeix el text al terminal com a referència de debug:

```
JavaScript

button = Gtk::Button.new(label: "Mostra el text")
button.signal_connect "clicked" do
  buffer = textview.buffer
  text = buffer.text
  lcd_print(lcd, text)
  puts "#{text}"
end
```

8. **Organització dels Widgets en una Caixa Vertical (`VBox`)**:

Els elements (`ScrolledWindow` amb `TextView` i el botó) s'agrupen en una `VBox` per organitzar-los en una disposició vertical. Això proporciona una estructura de disseny senzilla però efectiva per l'aplicació:

```
JavaScript
vbox = Gtk::Box.new(:vertical, 5)
vbox.pack_start(scrolled_window, expand: true, fill: true, padding:
0)
vbox.pack_start(button, expand: false, fill: false, padding: 10)
```

9. **Finalització i Execució de l'Aplicació**:

La `VBox` s'afegeix a la finestra principal i es configura l'event de tancament per finalitzar el programa amb `Gtk.main_quit`. Finalment, es mostra la finestra amb `show_all` i s'inicia el bucle principal de GTK amb `Gtk.main`:

```
JavaScript
window.add(vbox)
window.signal_connect("destroy") { Gtk.main_quit }
window.show_all
Gtk.main
```

Conclusió

Amb aquesta estructura, el codi compleix amb el requisit de mostrar text multilínia en una pantalla LCD utilitzant una interfície gràfica desenvolupada amb Ruby i GTK3. Les funcionalitats principals es compleixen amb senzillesa, utilitzant una estructura de disseny neta i una integració directa amb la pantalla LCD mitjançant I2C.