Puzle 2 - LCD

1. Pas previ: adaptació del codi del puzle 1 per importar-lo en el puzle 2

Per poder fer servir el codi que hem fet al puzle 1 en el puzle 2, cal crear un mòdul amb una funció i després importar-lo al nou codi. Llavors, passem d'aquest codi (que ja complia amb detalls addicionals que es van demanar per separar el codi per canvis de línia "\n" i truncar cada una d'elles a 20 caràcters):

```
import sys
from RPLCD.i2c import CharLCD

lcd = CharLCD('PCF8574', 0x27, cols=20, rows=4)

user_text = sys.stdin.read().strip().split("\n")[:4] # Llegir i dividir el text
for i, line in enumerate(user_text): # Escriure les linies al LCD
    lcd.cursor_pos = (i, 0)
    lcd.write_string(line[:20]) # Escriu cada linia truncada a 20 caracters
```

A aquest nou codi:

```
import sys
from RPLCD.i2c import CharLCD

lcd = CharLCD('PCF8574', 0x27, cols=20, rows=4)

def mostrar_texto_lcd(texto):
    user_text = sys.stdin.read().strip().split("\n")[:4] # Llegir i dividir el text
    for i, line in enumerate(user_text): # Escriure les linies al LCD
        lcd.cursor_pos = (i, 0)
        lcd.write_string(line[:20]) # Escriu cada linia truncada a 20 caracters
```

Amb això, podem reutilitzar el codi del puzle 1 en el puzle 2 de forma senzilla, important la funció `mostrar_text_lcd` i aprofitant la seva lògica per mostrar text a l'LCD.

2. Instal·lació de PyGObject

Per desenvolupar un entorn gràfic d'execució del programa, es requereix la biblioteca PyGObject, que ens permet treballar amb GTK en Python. Per instal·lar PyGObject, un requisit és tenir prèviament instal·lat cairo. L'instal·lem amb el següent comandament al terminal:

```
sudo apt install -y libcairo2 libcairo2-dev libgirepository1.0-dev
                    gir1.2-gtk-3.0 python3-gi python3-gi-cairo
```

Després, ja es pot instal·lar PyGObject amb:

```
pip install PyGObject
```

3. Implementació de la interfície gràfica amb GTK

L'objectiu del puzle 2 és fer un programa que ens demani gràficament el que abans fèiem des del terminal, amb parell de detalls. S'ha dissenyat una interfície senzilla que permet escriure un text amb un botó per enviar el text escrit al display.

Primer, per poder executar-lo correctament cal importar la llibreria `gi` i la funció `mostrar_text_lcd`, a més de definir que utilitzarem la versió 3.0 de GTK i importar els mòduls necessaris com `Gtk`, `GLib` i `Gdk` per gestionar la interfície gràfica i els events:

```
import gi
from lcd_codigo import mostrar_text_lcd  # Importams la funcio del Puzle 1

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib, Gdk
```

Després, s'ha creat una classe `EntryWindow`, que hereta de `Gtk.Window` amb quatre mètodes que defineixen la finestra principal (a la qual se li ha establert una mida de 285x235 píxels) i els seus elements.

- `__init__(self)`:

És la primera de les quatre funcions i és el constructor de la classe `EntryWindow`, que configura la interfície gràfica del programa. Inicialitzem la finestra amb un títol, una mida fixa i impedim que es pugui redimensionar (més endavant s'explica que és perquè el text que s'escrigui hi quadri correctament a l'ample i alçada de la finestra):

```
Gtk.Window.__init__(self, title="Puzzle 2 - Display LCD")
self.set_size_request(285, 235)
self.set_resizable(False)
```

Carreguem els estils CSS que aplicarem per personalitzar i millorar l'aparença el programa des d'un fitxer extern:

```
self.set_css("estils.css")
```

Afegim també un `TextView` on s'escriurà el text per enviar al display:

```
# TextView per escriure el text
self.entry = Gtk.TextView()
self.entry.set_wrap_mode(Gtk.WrapMode.WORD)
self.entry.set_size_request(-1, 100)
self.entry.get_style_context().add_class("text-entry")
self.entry_buffer = self.entry.get_buffer()
vbox.pack_start(self.entry, True, True, 0)
```

Creem un contenidor vertical amb espai de 10 píxels entre elements, li apliquem un estil CSS i l'afegim a la finestra:

```
vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
vbox.get_style_context().add_class("main-container")
self.add(vbox)
```

Afegim una etiqueta amb instruccions per a l'usuari i li apliquem un estil CSS per definir-ne l'aparença:

```
label = Gtk.Label(label="Maxim: 4 files i 20 caracters")
label.get_style_context().add_class("title-label")
vbox.pack_start(label, False, False, 0)
```

Creem un quadre de text, configurem l'ajust de paraules, li apliquem un estil CSS, definim un mida fixa perquè hi càpiguen exactament 4 línies i 20 caràcters per línia amb una font monospaced i l'afegim al contenidor:

```
self.entry = Gtk.TextView()
self.entry.set_wrap_mode(Gtk.WrapMode.WORD)
self.entry.get_style_context().add_class("text-entry")
self.entry_buffer = self.entry.get_buffer()
self.entry.set_size_request(249, 109)
vbox.pack_start(self.entry, True, True, 0)
```

Afegim un boto per enviar el text, el connectem a una funció, li apliquem un estil CSS i l'afegim al contenidor:

```
self.button = Gtk.Button(label="Enviar al LCD")
self.button.connect("clicked", self.on_button_clicked)
self.button.get_style_context().add_class("display-button")
vbox.pack_start(self.button, False, False, 0)
```

Finalment, connectem un event al buffer del quadre de text per limitar el text que l'usuari pot escriure:

```
self.entry_buffer.connect("changed", self.on_text_changed)
```

- `on_button_clicked(self, widget)`:

Aquesta funció s'executa quan l'usuari prem el botó "Enviar al LCD" de la interfície i la seva finalitat es obtenir el text que l'usuari ha escrit en el quadre de text en aquell moment i enviar-lo al display LCD perquè es mostri fent servir la funció `mostrar_text_lcd`, que vam fer en el puzle 1:

```
def on_button_clicked(self, widget):
    """Obte el texto del TextView i lenvia al LCD."""
    text = self.entry_buffer.get_text(self.entry_buffer.get_start_iter(),
                                      self.entry_buffer.get_end_iter(), True)
    mostrar_text_lcd(text)  # Cridem a la funcio del puzle1 per mostrar el text
    print("Text enviat al LCD:\n", texto) # Mostra el text enviat tambe al terminal
```

- `on_text_changed(self, buffer)`:

Aquest mètode es cridat automàticament cada vegada que el contingut del quadre de text canvia, ja sigui perquè l'usuari escriu, esborra o modifica el text. La seva tasca principal es assegurar que el text no superi els límits establerts: un màxim de 4 files i 20 caràcters per fila. Per evitar problemes com bucles infinits, primer bloqueja l'event que activa aquesta funció. Després, obté el text actual, el divideix en línies, i aplica les restriccions necessàries. Si el text modificat es diferent del text original, actualitza el quadre de text amb la versió limitada. Finalment, desbloqueja l'event perquè pugui seguir detectant futurs canvis:

```
def on_text_changed(self, buffer):
    """Limita el text a 4 filas i 20 caracters per fila"""
    buffer.handler_block_by_func(self.on_text_changed)

    text = buffer.get_text(buffer.get_bounds()[0], buffer.get_bounds()[1], True)

    linies = text.split("\n")
    if len(linies) > 4:
        linies = linies[:4]

    linies = [linia[:20] for linia in linias]
    text_limitat = "\n".join(linies)

    if text != text_limitat:
        buffer.set_text(text_limitat)

    buffer.handler_unblock_by_func(self.on_text_changed)
```

- `set_css(self, css_file)`:

Aquest mètode s'encarrega de carregar i aplicar els estils CSS definits en un fitxer extern (estils.css) per personalitzar l'aparença de la interfície. Crea un proveïdor d'estils CSS (`CssProvider`) i llegeix el fitxer CSS. Un cop carregat el contingut del fitxer, obté la pantalla actual on es mostraran els estils i crea un context d'estils. Finalment, assigna el proveïdor d'estils a la pantalla, assegurant que els estils definits en el fitxer CSS s'apliquin a tots els elements de la interfície gràfica:

```
def set_css(self, css_file):
    """Carrega i aplica estils CSS desde un fitxer"""
    css_provider = Gtk.CssProvider()

    with open(css_file, "rb") as css:
        css_provider.load_from_data(css.read())

    screen = Gdk.Screen.get_default()
    style_context = Gtk.StyleContext()
    style_context.add_provider_for_screen(screen, css_provider,
                                          Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION)
```

Amb això, les funcions del programa ja estan acabades i per iniciar l'aplicació creem una instancia de la finestra, connectem l'event de tancament per aturar el programa, mostrem tots els elements visuals i iniciem el bucle principal de `GTK`:

```
# Iniciem la aplicacio
win = EntryWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

4. Aplicació d'estils amb CSS

Per millorar l'aparença visual del programa, hem utilitzat estils CSS que defineixen els marges, colors, fonts i efectes dels elements de la interfície.

- `.main-container`:

Aquest estil defineix els marges del contenidor principal de la interfície, que es el layout vertical que conté tots els elements:

```
/* Defineix els marges del layout principal */
.main-container {
    margin-top: 15px;
    margin-bottom: 15px;
    margin-left: 15px;
    margin-right: 15px;
}
```

- `.title-label`:

Aquest estil configura l'aparença de l'etiqueta que mostra les instruccions a la part superior de la finestra:

```
/* Configura laparença de letiqueta dinstruccions */
.title-label {
    font-size: 20px;
    font-weight: bold;
    color: #4A4A4A;
}
```

- `.text-entry`:

Aquest estil defineix laparença i el comportament del quadre de text on l'usuari escriu el missatge per enviar al LCD:

```
/* Defineix laparença del quadre de text */
.text-entry {
    font-family: monospace;
    font-size: 20px;
    padding: 5px;
    border: 1px solid #A0A0A0;
    background-color: #FAFAFA;
    min-width: 20em; /* ample suficient per 20 caracteres */
    min-height: 4em; /* alcada suficient per 4 linias */
    border-radius: 8px;
}
```

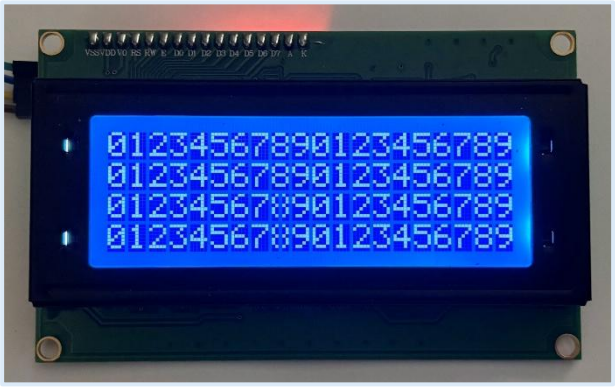
- `.display-button` i `.display-button:hover`:

Aquest estil configura laparença del boto "Enviar al LCD" i defineix com a detall un efecte quan l'usuari passa el ratolí per sobre i unes vores rodones al botó.

```
/* Configura laparença del boto i lefecte al passar-hi el ratolí */
.display-button {
    font-size: 20px;
    font-weight: bold;
    background-color: #4CAF50;
    color: white;
    padding: 10px;
    border-radius: 8px;
}
.display-button:hover {
    background-color: #45A049;
}
```

5. Resultat final

El resultat final de tot aquest puzle, és poder veure una finestra com aquesta:



Veiem que és un programa correcte, funcional, que compleix amb seva finalitat i té en compte detalls com que la finestra no s'ampliï quan s'excedeixen els 20 caràcters a cada línia o 4 línies en total i que utilitza una font monospaced per veure més exactament com es veurà el text al display quan es premi el botó d'enviar.