Ivan Rodriguez Canals

PBE - Second Puzzle Report

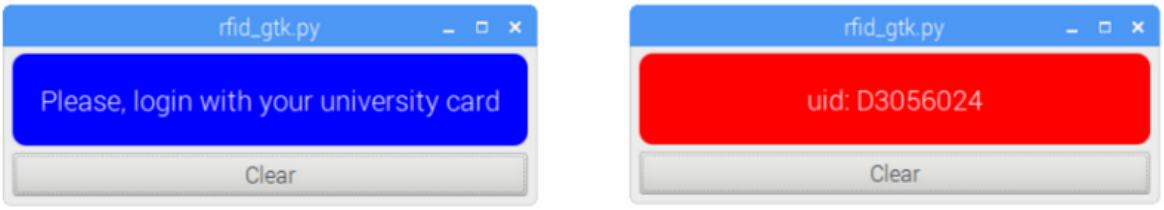
Main objective + library

The main objective of the second puzzle is to program a graphic version of the first puzzle previously created. In order to do this, we will need to use this library:

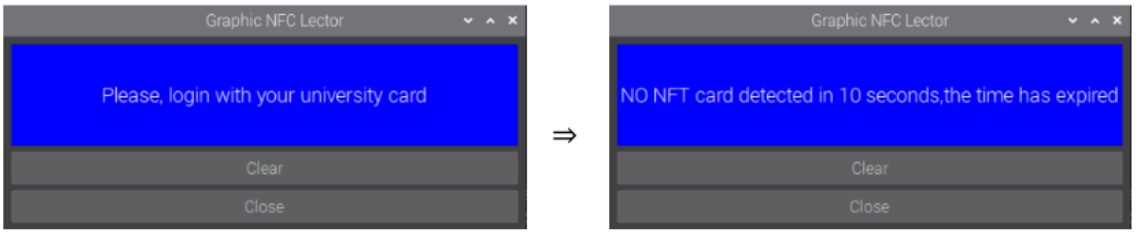
-PyGObject (GTK version 3.0)

The output should be a window with a label asking the student to log in with his respective UPC target or RF target and a button to clear. In my case, I added a button to quit. Once the student has approached the target to the NFC lector, its respective UID will pop up in the middle of the label. Apart from that, I also added a message that will appear if no card was detected (UID = null).

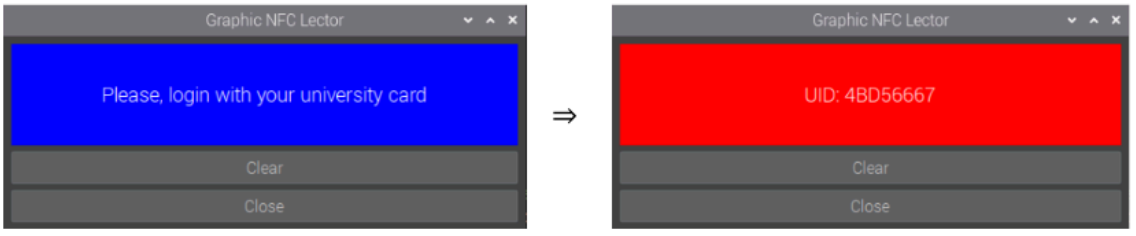
Professor example of implementation



My implementation



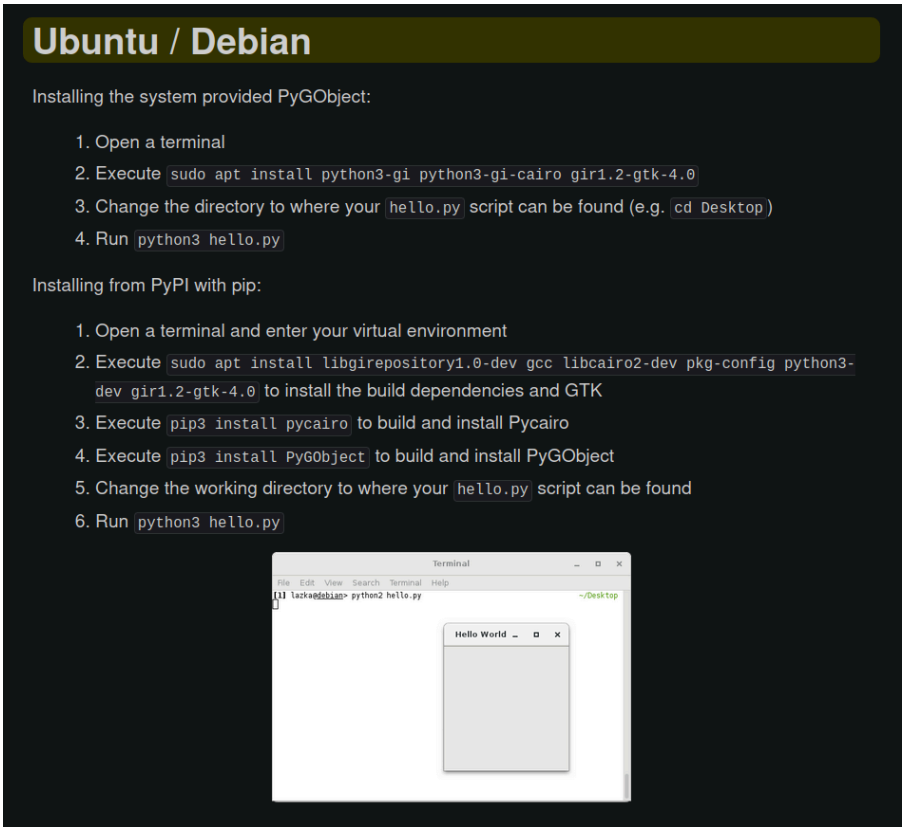
⇑ If the NFC does not detect a target ⇑



⇑ If the NFC detects a target ⇑

Installation and implementation process

To install the library (in a virtual environment), I followed the next steps, which are in the web page of the library in the Getting Started apartment.



Once I have proven that the installation was successful, running *pytohn3 hello.py* in the terminal, I started looking for examples to implement the requested task. In the process, I found a very interesting '**PyGObject-Tutorial**' on GitHub, which is also available in web page format. Tutorial's web site . This tutorial helped me to create the label and the buttons.

Regarding the question that the read_uid is blocking, and graphical environments are not blocking, I used the next documentation applied to PyGObject Threads & Concurrency. With the examples in the tutorial/discussion, I managed to implement the program so it doesn't block executing read_uid in an auxiliary thread.

CODE with comments

second_puzzle.py > ...

```python
import gi
import threading
from primer_puzzle_adafruit import RfidReader #import the class of the card reader
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GObject, Gdk

class GraphicPuzzle(Gtk.Window):

    def __init__(self): # All the initializations
        super().__init__(title="Graphic NFC Lector")

        self.set_border_width(8) # Select the border width
        self.rfid_reader = RfidReader()  # Initialize the lector

        # Create a box , set the orientation and the separation from the buttons
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)

        # Create a label
        self.label = Gtk.Label()
        self.label.set_size_request(500,100) #label size
        #size and color of the text
        self.label.set_markup('<span size="15000" foreground="white">Please, login with your university card</span>')
        self.label.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(0, 0, 1, 1))  # blue background
        vbox.pack_start(self.label, True, True, 0) #starts the label

        # Clear button
        clear_button = Gtk.Button.new_with_mnemonic("Clear")
        clear_button.connect("clicked", self.on_clear_clicked)
        vbox.pack_start(clear_button, True, True, 0)

        # Close buttons
        close_button = Gtk.Button.new_with_mnemonic("Close")
        close_button.connect("clicked", self.on_close_clicked)
        vbox.pack_start(close_button, True, True, 0)

        # Initializes the thread that reads the UID of the card
        self.uid_thread()

    def uid_thread(self):
        # Runs the lecture of the card read_uid form another thread
        thread = threading.Thread(target=self.read_uid, daemon=True)
        thread.start()

    def read_uid(self):
        # Calls the method that reads the card (from first_puzzle)
        uid = self.rfid_reader.read_uid()

        #If the card is found, the label is updated
        # This way it wont freeze when you press the clear button
        if uid: # If UID is not null
            GObject.idle_add(self.update_label, f"UID: {uid}", "red")
        else: # If UID is null
            GObject.idle_add(self.update_label, "NO NFT card detected in 10 seconds,the time has expired", "blue")

    def update_label(self, text, color):
        # Updates the text and background of the label
        if color == "red":
            self.label.set_size_request(500,100)
            self.label.set_markup(f'<span size="15000" foreground="white">{text}</span>')
            self.label.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(1, 0, 0, 1))
        else:
            self.label.set_size_request(500,100)
            self.label.set_markup(f'<span size="15000" foreground="white">{text}</span>')
            self.label.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(0, 0, 1, 1))

    def on_clear_clicked(self, button):
        # Clearing the label and resetting the startup message
        self.label.set_size_request(500,100)
        self.label.set_markup('<span size="15000" foreground="white">Please, login with your university card</span>')
        self.label.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(0, 0, 1, 1))
        # Call the lecture process
        self.uid_thread()

    def on_close_clicked(self, button):
        print("Closing application")
        Gtk.main_quit()

# Create the window and start the graphical interface
win = GraphicPuzzle()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```