Puzzle 2 - Python

El meu programa pel puzzle 2 es el següent:

```
import gi
import threading
import nfc
import signal
from Rfid_Lector import Rfid_Lector
from gi.repository import Gtk, GLib, Gdk

gi.require_version('Gtk', '3.0')

class RfidApp(Gtk.Window):
	def __init__(self):
		super().__init__(title = "Lector RFID")
		self.set_default_size(400, 200)
		self.set_border_width(20)

		distribucio = Gtk.Box(orientation = Gtk.Orientation.VERTICAL, spacing = 6)
		self.add(distribucio)

		self.msg = Gtk.Label(label = "Apropeu la targeta siusplau")
		self.msg.set_size_request(300, 100)
		self.msg.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(0, 0, 1, 1))
		self.msg.override_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(1, 1, 1, 1)
		distribucio.pack_start(self.msg, True, True, 0)

		self.buto = Gtk.Button(label = "Clear")
		self.buto.connect("clicked", self.clear)
		distribucio.pack_start(self.buto, True, True, 0)

		self.rfid_lector = Rfid_Lector()

		self.lect_thread = threading.Thread(target=self.lectura_rfid)
		self.lect_thread.daemon = True
		self.lect_thread.start()

	def clear(self, widget):
		self.msg.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(0, 0, 1, 1))
		self.msg.set_text("Apropeu la targeta siusplau")

	def lectura_rfid(self):
		while True:
			uid = self.rfid_lector.read_uid()
			if uid:
				GLib.idle_add(self.update_msg_uid, uid.upper())
			else:
				threading.Event().wait(1)


	def update_msg_uid(self, uid_hex):
		self.msg.override_background_color(Gtk.StateFlags.NORMAL, Gdk.RGBA(1, 0, 0, 1))
		self.msg.set_text("uid: " + uid_hex)

	def on_close(self, *args):
		self.rfid_lector.disconnect_reader()
```

```
                Gtk.main_quit()

if __name__ == "__main__":
        a = RfidApp()
        a.connect("destroy", Gtk.main_quit)
        a.show_all()
        Gtk.main()
```

Aquest programa implementa la nova funció RfidApp, la qual crea una finestra flotant que exposi la UID de la targeta al llegirla pel lector.
Per tal de fer això disposa de una zona de text que varia de color depenent de si ha detectat la targeta i un botó per tal de netejar la informació llegida i poder tornar a llegir un altre informació.

Com podem observar aquest programa està utilitzant com una llibreria el programa del puzzle anterior Rfid_Lector, en el qual també hi ha hagut algun canvi, degut a problemes que m'he trobat pel cami (Pero d'això ja en parlarem més tard).

La funció arranca iniciant els valors necessaris per la creació de la finestra flotant, com poden ser: el titol, la mida de la finestra i dels bordes d'aquesta, una zona de text on escriure un missatge inicial o les dades amb el color pertinent, un botó per eliminar les dades obtingudes, la crida del programa anterior com a llibreria i finalment la creació del thread.

El thread en aquest programa funciona com a daemon i serveix per actualitzar les dades a la finestra flotant, es a dir, com que el programa de la finestra flotant no pot mirar constantment si esta revent la targeta, creem un thread que ho faci i utilitzant funcions de update que li passi el missatge del thread a la finestra flotant.

La funció clear es la utilitzada cada cop que utilitzem el botó de clear, per reiniciar el missatge i el color als inicials.

La funció de lectura de la uid, utilitza la funció del Puzzle passat, tot hi que aquest ha rebut un petit canvi. Un cop agafat el valor, mirem si aquest es valid, en cas de no haver llegit res, s'espera un segon, així podem disminuir el trafic exagerat que sobrecalentava la CPU. En cas de haver llegit un valor valid, aquest crida la funció de update del missatge a la finestra flotant.

La funció de update del missatge de la finestra flotant, canvia el color i el missatge de aquesta.

Finalment podem trobar el main, on definim una variable amb la clase RfidApp.

Aqui podem observar el programa del Puzzle anterior amb els canvis pertinents:

```
import nfc

class Rfid_Lector:
        def __init__(self):
                self.clf = None

        def connect_reader(self):
                if self.clf is None:
                        try:
                                self.clf = nfc.ContactlessFrontend('usb')
                        except Exception as e:
                                print("Error al llegir la targeta: " + str(e))
                                self.clf = None
```

```
	def read_uid(self):
		self.connect_reader()
		if self.clf:
			try:
				tag = self.clf.connect(rdwr={'on-connect': lambda tag: False})
				uid = tag.identifier.hex()
				return uid
			except Exception as e:
				print("Error al llegir la targeta NFC: "+ str(e))
				return None
		else:
			print("Lector NFC no disponible o desconectat")
			return None

	def disconnect_reader(self):
		if self.clf:
			self.clf.close()
			self.clf = None

if __name__ =="__main__":
	rf = Rfid_Lector()
	print("Esperant targeta NFC...")
	uid = rf.read_uid()
	if uid:
		print("UID de la targeta: "+ uid.upper())
	else:
		print("No s'ha pogut llegir la targeta correctament")
```

En aquest nou programa Rfid_Lector, s'ha modificat les funcions ja creades i s'han implementat noves funció connect_reader() i disconnect_reader(), les quals obliguen a tenir un control de quan es conecta i quan es desconecta el Lector NFC.

M'he vist obligat a fer aquests canvis, degut a que quan el programa principal executava el thread amb la lectura de la uid aquest mai parava, tot hi que el programa principal finalitzes aquest quedava actiu, el que feia que el Lector NFC quedes en un thread, i a la posterior execució no pogues trobar el component.