Wizix

Zeste de savoir

Des interfaces graphiques en Python et GTK

12 août 2019

Table des matières

Les interfaces graphiques vous en voyez tout le temps, sans même y penser. En effet, ce sont ces fenêtres avec des boutons, des menus, des champs de texte... Et si vous appreniez comment en faire ? Vous allez voir que ce n'est pas aussi compliqué que ce que vous pouvez le penser, et même si la console permet de faire des choses vraiment très poussé en terme d'interface, un utilisateur lambda sera bien plus à l'aise avec une interface graphique ! Vous êtes prêt à traverser la fenêtre ?

Vous êtes alors le/la bienvenue dans ce cours. Je vais vous montrer comment utiliser GTK+ 3 , à travers un binding[1] python de GObject sur lequel se base GTK+ 3 (oui c'est un peu compliqué, mais je vous promet que ça fonctionne !), afin de créer votre propre interface graphique.

Pour suivre ce tutoriel, il vous sera évidemment conseillé d'avoir une bonne connaissance de Python. La programmation d'interface graphique apporte son lot de problème que l'on avait pas avec la console. Celle-ci effectuait une suite d'action dans l'ordre définie par vous, le programmeur. Une interface graphique exécute les instructions dans **l'ordre que l'utilisateur choisi**. C'est à vous de penser à tous les cas.

Ça peut paraître effrayant comme ça, mais ne vous inquiétez pas, PyGObject nous simplifie grandement la vie. On commence ?

1. Un binding est le portage d'une bibliothèque écrite avec un certain langage vers un nouveau langage. En effet, GObject est écrite en C mais nous allons l'utiliser avec Python.

Première partie

Découverte

Vous apprendrez ici les bases de la programmation événementielle, c'est à dire basé sur des événements. Un clique sur un bouton, un scroll dans une fenêtre, un clique droit dans une zone de texte sont tous des événements.

Mais avant nous verrons comment installer tout ce qu'il vous faut pour commencer. PyGObject (que je raccourcirais parfois par son *petit nom*, Pygi) est multiplate-forme, c'est à dire qu'il fonctionne sans problème sous Linux, Windows et Mac OS X. Je n'ai cependant jamais réussis à l'installer sous cette dernière plate-forme.

1. Installation

Avant de commencer à créer vos GUI (*Graphical User Interface*), il va falloir installer PyGObject et ses dépendances. PyGObject est un module Python qui donne l'accès aux développeurs aux bibliothèques basées sur GObject comme, dans le cas qui nous intéresse, GTK+. On va donc pouvoir utiliser GTK+ avec Python !

Voici les dépendances requises pour l'utiliser :

— GTK+ 3
— Python 3.1 au minimum
— gobject-introspection

1.1. Sous Linux

On est assez chanceux sous Linux, chez nous c'est hyper simple.
Si vous utilisez une distribution récente, vous avez normalement toutes les dépendances déjà présentes sur votre ordinateur pour commencer à coder. Il ne vous manquera juste qu'un paquet. Installez-le avec :

`apt-get`

```
1 sudo apt-get install python3-gi
```

`pacman`

```
1 sudo pacman -S python-gobject
```

Et voilà !

1.2. Sous Windows

Toutes les images de cette partie ont été prises par WinXaito que je remercie.

On va commencer par télécharger l'installeur. Rendez-vous à cette adresse et téléchargez la dernière version en date de PyGi. Au moment où j'écris ce tutoriel, l'installeur se nomme *pygi-aio-3.14.0_rev22-setup.exe*.

Ouvrez-le. Une fenêtre comme celle-ci devrait apparaître :

FIGURE 1.1.

Cliquez sur **Ok** pour continuer. PyGObject vous demande ensuite d'indiquer le chemin de votre installation Python 3.

FIGURE 1.2.

Cliquez sur **Add path**. Cette fenêtre fait alors son apparition.

FIGURE 1.3.

Déplacez-vous dans votre arborescence afin de sélectionner l'emplacement de Python3. Habituellement c'est C :\Python3x. Cliquez ensuite sur **Ok**.

Dans la fenêtre suivante, vérifiez que la case soit bien cochée.

FIGURE 1.4.

Cliquez sur **Ok**. (Oui oui on a presque terminé, ne vous inquiétez pas !)

Maintenant vous allez choisir quels paquets vous souhaitez installer. En effet, comme dit dans l'introduction, PyGObject rassemble toute une collection de paquets dont GTK+ 3.

Nous aurons besoin de (ne prêtez pas attention aux numéros de version) :

— Base packages
— GTK+ 3.14.15
— GtkGLExt 2.99.0git
— GTKSourceView 3.14.4

Sélectionnez-les dans la liste.

FIGURE 1.5.

Une fois fini, cliquez sur **Ok**. L'installation se termine comme une installation basique. Vous avez enfin fini !

1.3. Vérification de l'installation

Allez les Windowsiens, on se dépêche ! Les Linuxiens vous attendent depuis des heures !

Bon, c'est bien beau d'avoir tout installé, mais est-ce que ça fonctionne ? Vérifions-le. Ouvrez votre interpréteur Python et importez GTK+ :

```
1 Python 3.4.3 (default, Mar 26 2015, 22:03:40)
2 [GCC 4.9.2] on linux
3 Type "help", "copyright", "credits" or "license" for more
      information.
4
5 >>> from gi.repository import Gtk
6 >>>
```

Si vous avez une erreur, n'hésitez pas à me la signaler avec un post sur le en utilisant les tags *[python]* et *[gtk]*. Vous ne pouvez pas continuer le tutoriel avec.

1.4. Comment lire la documentation

Je vous promet que vous avez tout intérêt à bien vous entendre avec cette documentation. En effet, vous allez y passer du temps car il est juste impossible de retenir toutes les méthodes offertes par GTK+. Je me dois donc de vous montrer comment la lire : rien de bien compliqué je vous rassure, elle est vraiment très bien faite et complète.

Bien, rendez-vous sur ce lien : http ://lazka.github.io/pgi-docs/Gtk-3.0/ . Vous l'aurez compris, c'est de cette documentation que l'on va parler ici.

Descendez la page et arrêtez vous à la section API . Comme vous pouvez le voir, la documentation se décompose en dix grandes parties :

Partie	Description
Functions	Partie que nous n'utiliserons pas, elle rassemble les différentes fonctions pour contrôler ou voir l'état de GTK.
Callbacks	De même, elle vous sera inutile. Elle rassemble les callbacks (mot que nous étudierons après) des différents widgets (pareil) de GTK.
Interfaces	Les interfaces de GTK.
Classes	**LA** partie qui nous intéresses le plus. Cette partie référence tous les widgets que vous allez pouvoir utiliser pour construire votre interface. J'en ai compté à peu près 258. Sachant que chaque widget à ses propres méthodes et signaux, j'espère que vous comprenez désormais l'intérêt de cette documentation.
Hierarchy	Partie un peu spéciale, car contrairement aux autres, elle ne vous renseigne pas sur l'utilité de telle ou telle chose mais sur comment est organisé GTK.
Structures	Documente tous les types de données créés par GTK. Nous utiliserons seulement le `Gtk.TreeIter` dans la seconde partie de ce tutoriel.
Flags	Les différents drapeaux utilisés par les widgets pour décrire leur état. Je n'en ai personnellement jamais utilisé un seul.
Enums	Rassemble les différentes énumérations de GTK. Par exemple, si vous souhaitez centrer un widget, vous allez utiliser `Gtk.Align.CENTER`
Constants	Les constantes de GTK.
Symbol Mapping	Si vous avez vu une fonction appelée en C, utilisez ces tableaux pour savoir comment l'appeler avec Python (une recherche sur la page est beaucoup plus efficace que de chercher à la main).

Je vais vous détailler le fonctionnement de la partie *classes*. Tous les widgets sont classés par ordre alphabétique. Je vous laisse trouver la classe `Gtk.Button` qui permet de créer des boutons. C'est bon, vous y êtes ?

Au tout début de la page, vous avez le droit à l'arbre présentant de quelles classes hérite un bouton. Comme vous pouvez le voir, c'est complexe. Les classes en bleu sont des interfaces tandis que celles en gris peuvent être instanciées.

Figure 1.6. – Arbre généalogique d'un bouton de GTK+ 3

Dessous vous avez un exemple et toutes les *subclasses* c'est à dire les classes qui hérites de `Gtk.Button`. Vous trouvez ensuite une liste complète de toutes les méthodes, cliquez sur une pour voir en détail ce qu'elle fait, ce qu'elle prend comme argument, ce qu'elle retourne.

Viennent ensuite les méthodes virtuelles. C'est grâce à elles que vous allez pouvoir émettre des signaux manuellement (ne vous inquiétez pas si vous ne savez pas de quoi je parle, nous verrons ça dans un prochain chapitre).

Les propriétés du widgets sont en-dessous. Chaque propriété peut être changée soit grâce à une méthode soit manuellement. Par exemple :

```
1 button.set_label('Je suis le texte dans le bouton : un label quoi')
2 # sera équivalent à
3 button.props.label =
      'Je suis le texte dans le bouton : un label quoi'
```

Listing 1 – Changement du texte d'un bouton

Je vous conseille cependant de passer par les setteurs/getteurs.
Vous trouverez aussi une section sur les différents signaux que peut émettre un bouton.
Puis vient ensuite la description en profondeur de chaque méthode, chaque constructeur, chaque signal... Comme je vous l'ai dit, la documentation est super complète et précise, profitez-en !

C'bon vous êtes fin prêt ! Vous allez pouvoir réaliser votre toute première fenêtre. Si ce n'est pas excitant ça !

2. Une première fenêtre, le Hello world

L'heure est arrivée. Nous allons faire notre première fenêtre. Voici à quoi elle va ressembler :

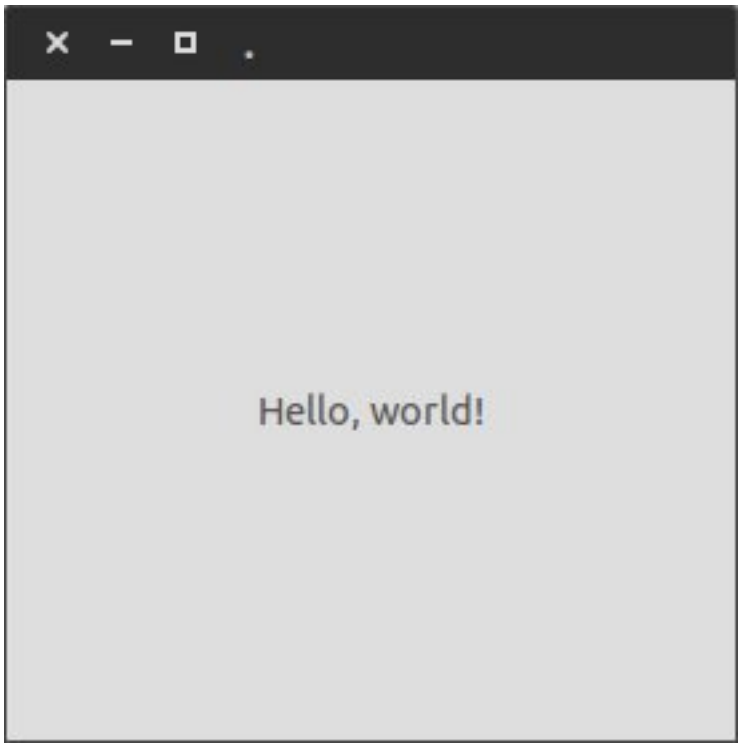

FIGURE 2.1. – Votre toute première fenêtre GTK+ 3 et Python3

Oui, okay elle est assez simpliste et je vois déjà vos airs déçu ! Mais ne vous en faites pas, les choses vont rapidement devenir intéressantes.
On commence ?

2.1. Avec l'interpréteur

Bien, ouvrez votre interpréteur Python3 et importez-y `Gtk` de GObject :

```
1 Python 3.4.3 (default, Mar 26 2015, 22:03:40)
2 [GCC 4.9.2] on linux
3 Type "help", "copyright", "credits" or "license" for more
      information.
4 >>> from gi.repository import Gtk
5 >>>
```

Nous allons commencer par créer une fenêtre qui va contenir nos widgets.
Un widget est un élément graphique. Cela peut être un bouton, une liste déroulante, un label (un texte), une image... Et même une fenêtre ! Pour en créer une, entrez ce qui suit :

```
>>> window = Gtk.Window()
```

Notre objet `window` est désormais une fenêtre GTK. Mais celle-ci est terriblement vide et ennuyante.
Remplissons-la un peu avec un label :

```
>>> label = Gtk.Label('Hello, world !')
>>> window.add(label)
```

Enfin, et le plus important pour vous j'imagine, affichons notre fenêtre :

```
>>> window.show_all()
>>> Gtk.main()
```

Sous vos yeux ébahis s'affiche une minuscule fenêtre. Approchez-vous d'un bord pour l'agrandir. Pour la fermer, cliquez sur la croix, comme une fenêtre normale. Cependant vous devrez fermer votre interpréteur Python avec un Ctrl+z.

2.2. La boucle GTK

J'aimerais revenir sur l'instruction `Gtk.main()`.
Une interface graphique est *simplement* une boucle géante qui attend patiemment que l'utilisateur fasse quelque chose. `Gtk.main()` lance cette boucle. Tant que celle-ci est lancée, votre programme tourne toujours.

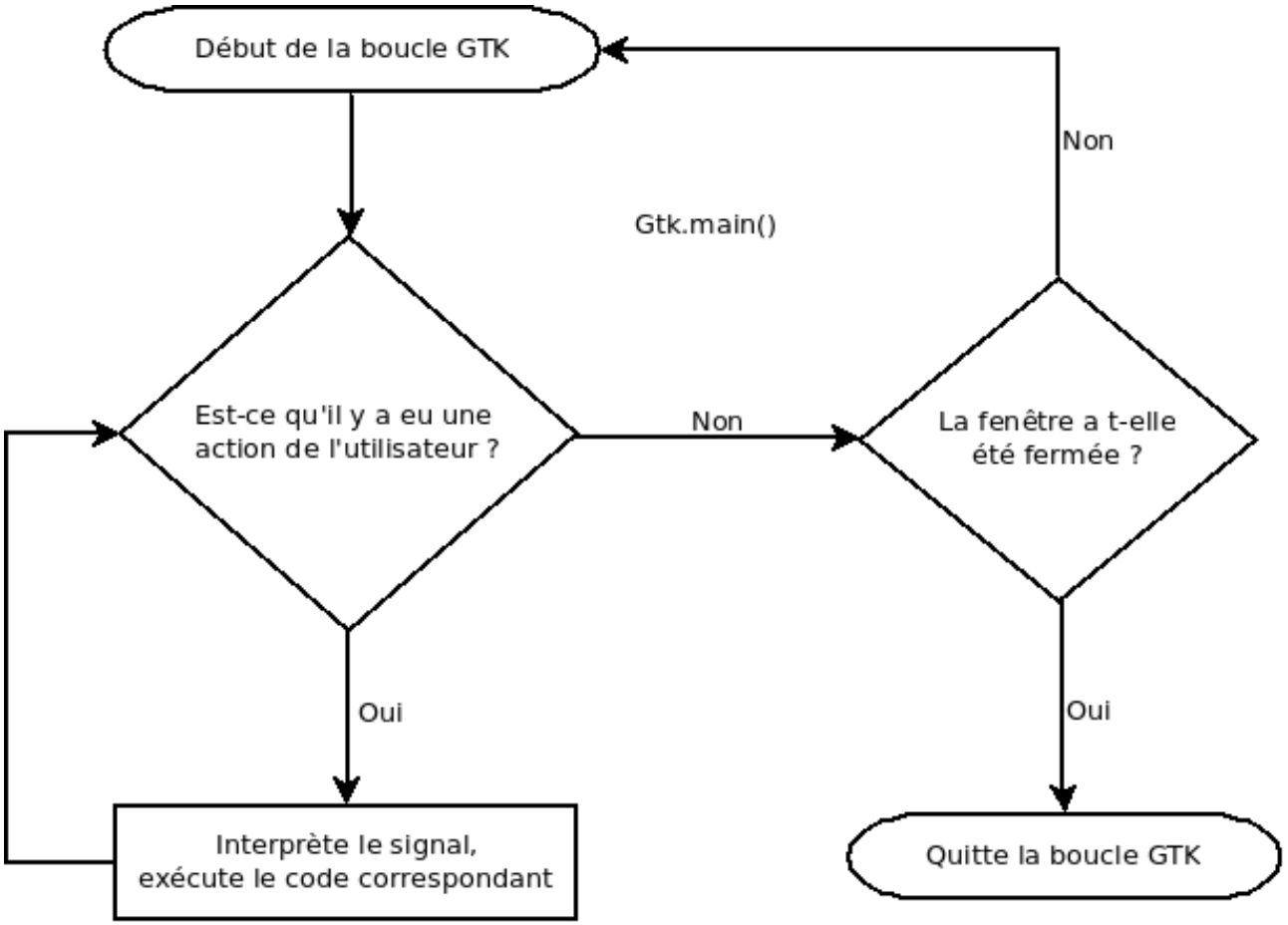

FIGURE 2.2. – Diagramme de la boucle Gtk.main

C'est pour ça que l'on a dû forcer l'interpréteur à se fermer, on n'a jamais indiqué à GTK+ de sortir de cette boucle.
Pour fermer cette boucle, nous devons utiliser la méthode `Gtk.main_quit()`. Cependant, nous souhaitons appeler cette méthode seulement quand on ferme la fenêtre, c'est à dire quand on la supprime.
Nous allons donc devoir utiliser les événements, comme ceci :

```
1 >>> window.connect('delete-event', Gtk.main_quit)
```

Ici, nous indiquons que quand l'événement `delete-event` est appelé, en supprimant la fenêtre par exemple, on appelle la méthode `main_quit` de GTK.
Je vous expliquerais plus en détail dans le prochain chapitre comment utiliser ces événements et leur fonctionnement.
Un peu d'exercice, ça vous tente ?

2.3. Un peu d'exercice

Tout d'abord, voici ce que donne notre *Hello world* dans un fichier `hello_world.py` :

```python
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk

# On crée notre fenêtre principale
window = Gtk.Window()

# On crée également un label pour notre window
label = Gtk.Label('Hello, world!')

# On l'ajoute à la fenêtre
window.add(label)

# On indique que le si la fenêtre est supprimée, la boucle
    principale s'arrête
window.connect('delete-event', Gtk.main_quit)

# On affiche toute notre fenêtre
window.show_all()
# Et on lance la boucle principale
Gtk.main()
```

Nous obtenions ce résultat :

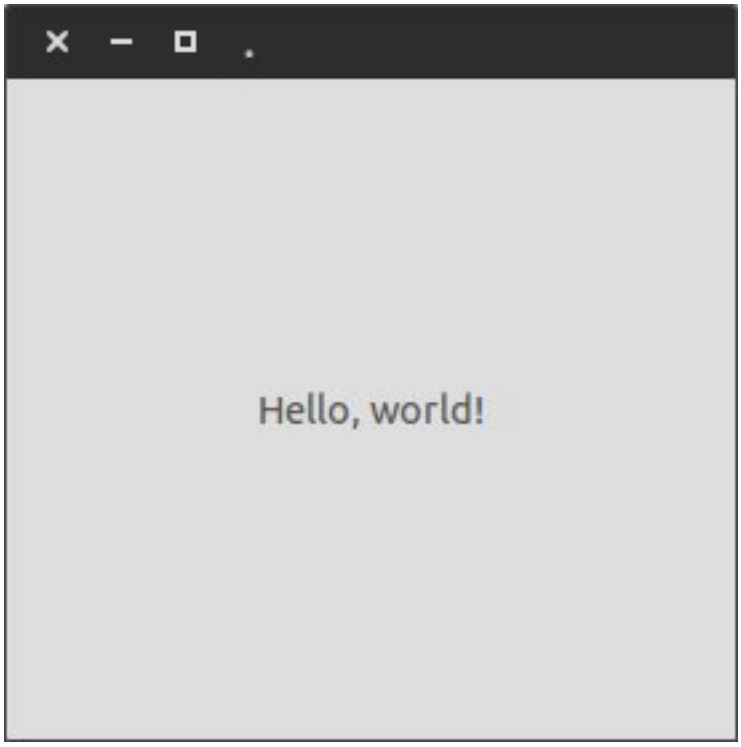

FIGURE 2.3. – La fenêtre Hello World

Voici donc quelques défis que je vous ai concocté :

— Créez une fenêtre qui afficherait `Bonjour les zesteux!`.

— Assignez un titre à votre fenêtre.
— Créez une fenêtre qui aurait un bouton avec comme texte `Bonjour les zesteux!`. Bien sûr, ce bouton ne fait rien pour le moment.

? Mais comment je fais pour réaliser ces deux derniers défis ? Tu ne nous as jamais montré comment faire ça !

Héhé, justement, c'est là tout le vice de mes défis. Si vous êtes là, c'est que vous êtes habitué de Python et que vous savez chercher dans une documentation. Voilà le lien de celle de PyGObject . Mon objectif est que soyez capable de vous débrouiller en autonomie en vous laissant chercher tout seul. C'est le meilleur moyen d'apprendre surtout que lorsque l'on crée des GUI, on passe notre temps à faire des aller-retours entre notre code et celle-ci.
Voici les solutions de ces trois défis. Je vous serais reconnaissant d'essayer avant de regarder les solutions.
Créer une fenêtre qui afficherait `Bonjour les zesteux!`. Ce premier défi était très simple, j'espère que vous l'avez réussi.

Contenu masqué n°1

C'est à partir de là que ça se complique.
Assigner un titre à notre fenêtre :

Contenu masqué n°2

La réponse se trouvait ici , il fallait fouiller dans la liste jusqu'à trouver la méthode `set_title()`.
Et enfin, créer une fenêtre qui aurait un bouton avec comme texte `Bonjour les zesteux!` :

Contenu masqué n°3

Encore une fois, il fallait chercher dans la documentation une classe qui permettrait de créer un bouton. En toute logique, la classe `Gtk.Button` correspondait exactement à ce que l'on voulait.

J'espère que tout c'est bien passé et que mes petits défis n'auront pas été trop douloureux. Ceux des chapitres suivants vont être bien plus simple maintenant que vous savez vous servir de la documentation.
La suite ? Comme promis, les événements.

Contenu masqué

Contenu masqué n°1

```python
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk

window = Gtk.Window()

label = Gtk.Label('Bonjour les zesteux!')
window.add(label)

window.connect('delete-event', Gtk.main_quit)

window.show_all()
Gtk.main()
```

Retourner au texte.

Contenu masqué n°2

```python
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk

window = Gtk.Window()
window.set_title('Un hello world')  # Fallait la trouver dans la
    doc celle-ci :p

label = Gtk.Label('Bonjour les zesteux!')
window.add(label)

window.connect('delete-event', Gtk.main_quit)

window.show_all()
Gtk.main()
```

Retourner au texte.

Contenu masqué n°3

```python
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk

window = Gtk.Window()
window.set_title('Un hello world')

button = Gtk.Button(label='Bonjour les zesteux!')
window.add(button)

window.connect('delete-event', Gtk.main_quit)

window.show_all()
Gtk.main()
```

Retourner au texte.

3. Les événements

Les événements sont un concept fondamental qu'il vous sera important de bien comprendre. Ce n'est pas compliqué, vous allez voir. On retrouve ce concept dans les autres bibliothèques graphiques comme Qt ou TKinter.
Nous apprendrons également comment faire un bouton et l'utiliser. Ce chapitre est donc fondamental, prenez votre temps pour le lire !

3.1. Présentation

Les événements, je vous le dis tout de suite, vous allez en manger ! Matin, midi et soir en n'omettant pas le goûter bien sûr.
Nous allons découvrir ensemble la programmation événementielle. C'est-à-dire que nous allons réagir aux actions de l'utilisateur. Un clique sur un bouton, la validation d'une zone de texte, etc. Ce sont des événements ! Vous imaginez bien que sans ça, votre programme serait d'un ennui... Toute la puissance de l'interface graphique se trouve là-dedans.
Vos programmes ne suivront désormais plus du tout le même plan. Je m'explique :

- Avant, vos programmes étaient dits linéaires : votre code s'exécutait dans l'ordre où vous l'aviez écrit (du haut vers le bas), vous étiez d'ailleurs assez limités par cette contrainte. De plus, l'utilisateur n'avait aucune liberté, il ne pouvait rien faire d'autre que ce que vous lui proposiez.
- Maintenant, c'est une toute autre façon de penser. Dans votre programme, vous définissez votre interface, les boutons et tout ça, on lance le tout, puis on passe la main à l'utilisateur. C'est lui qui choisi ce qu'il fait, quand, et comment.

Cette méthode de programmation est dite événementielle, elle suit le principe action → réaction.
L'objectif du jour est de savoir faire réagir notre application à la réception de ces **signaux**. Le principe des signaux est vraiment simple : lorsque l'utilisateur utilise votre **widget**, celui-ci annonce a qui veut entendre ce que vous lui avez fait, c'est ce que l'on appelle *un signal* (qui sera différent pour chaque action que le widget gère).
GTK s'occupe de tout, donc la partie technique ne nous regarde pas. Cependant il faut bien comprendre comment ça fonctionne.
Il n'y a qu'une seule chose à faire. On va utiliser une fonction qui dira à GTK :

- De quel widget on veut traiter un événement.
- Le signal (l'événement) que l'on veut traiter.
- Le nom d'une fonction (faite par vous), à appeler lorsque cet événement survient. Ce type de fonction est appelé fonction *callback*.

Cette partie n'a pas été écrite par ma propre plume. Elle se trouve originellement sur OpenClassroom et est publiée sous la licence CC-BY−SA. Je pense que je n'aurais pas pu vous expliquer aussi bien le concept des événements qu'eux.

En gros, on peut résumer avec un petit schéma :

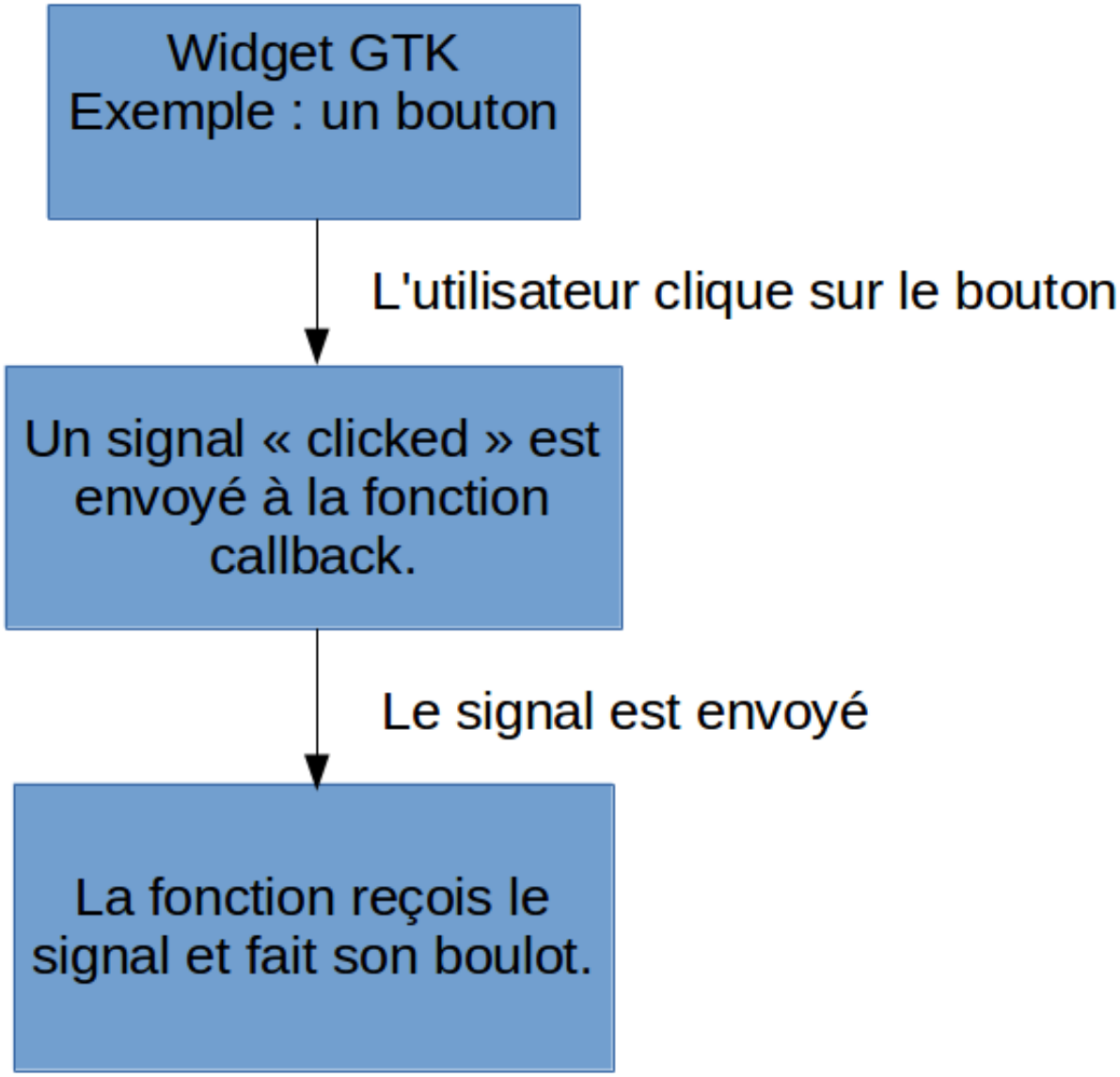

FIGURE 3.1. – La vie d'un événement

3.2. Premier contact

Reprenons notre fenêtre avec le bouton que vous avez (normalement) réalisée dans le chapitre précédant :

```
1 #!/usr/bin/env python3
2 # coding: utf-8
3
4 from gi.repository import Gtk
```

```
5
6  window = Gtk.Window()
7  window.set_title('Un hello world')
8
9  button = Gtk.Button(label='Dire bonjour')  # Je modifie juste le
       label du bouton
10 window.add(button)
11
12 window.connect('delete-event', Gtk.main_quit)
13
14 window.show_all()
15 Gtk.main()
```

Ce que je voudrais, c'est que quand je clique sur le bouton, la fenêtre me dise bonjour. Vous l'aurez deviné, il va falloir utiliser les événements. Voici ce que ça donne :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  # On créer une fonction toute simple
7  def say_hello(button):
8      ''' Fonctions qui dit bonjour quand on clique sur le bouton '''
9      print('Bonjour les zesteux !')
10
11 window = Gtk.Window()
12 window.set_title('Un hello world')
13
14 button = Gtk.Button(label='Dire bonjour')  # Je modifie juste le
       label du bouton
15 window.add(button)
16
17 # On connecte ce bouton à la fonction
18 button.connect('clicked', say_hello)
19
20 window.connect('delete-event', Gtk.main_quit)
21
22 window.show_all()
23 Gtk.main()
```

Normalement c'est là que vous me posez toutes vos questions. Bon là ça va être un peu compliqué. Je vais quand même essayer de répondre aux questions que vous pourriez vous poser.

3.3. La notion d'événement

Nous allons nous intéresser à cette ligne : `button.connect('clicked', say_hello)`. Dans GTK, chaque widget possède une méthode `connect`. Voici sa syntaxe :

```
1 handler_id = widget.connect('event', callback, data)
```

L'objet `widget` représente n'importe quel widget de GTK. Pour nous, ce sera un bouton. Cette méthode prend deux arguments obligatoires et un facultatif. Le premier est le nom le l'événement. Par exemple, si vous utilisez un bouton, vous voulez qu'un signal soit émis quand l'utilisateur clique dessus, vous allez utiliser `'clicked'`. Chaque widget a ses propres événements.
Le second est la fonction dite *callback*. C'est à dire que vous allez passer en argument une fonction qui va réagir au signal. Le widget est automatiquement passé en argument.
Pour finir, vous allez mettre en dernier argument les arguments que vous souhaitez passer à votre fonction callback. Nous étudierons ce cas plus tard, ne vous inquiétez pas.
La fonction `connect` retourne un nombre qui identifie cette connexion. Celui-ci peut être utile si vous avez besoin de déconnecter le widget de cet événement ou de lui attribuer un autre callback.
Reprenons notre ligne qui va tout à coup vous paraître bien plus simple :

```
1 button.connect('clicked', say_hello)
```

Nous demandons à GTK+ de connecter notre `button` à la fonction callback `say_hello` quand l'événement `'clicked'` est émis.

3.4. Entrainons-nous

Quoi ? Vous pensiez réellement que j'allais vous laisser tranquille cette fois ? Certainement pas !
Les événements étant un concept phare de GTK+, je vous conseille vraiment de réaliser ces petits défis (sans tricher bien sûr). Ce que je vous propose :

- Un bouton qui change de label quand on clique dessus
- Un bouton qui quitte le programme
- Un bouton qui créerait une nouvelle fenêtre contenant un autre bouton qui ferme tout (plus difficile celui-ci !)

Allez, à vos éditeurs !
Pour le premier défi, vous avez sûrement utilisé la documentation afin de savoir comment changer le label d'un bouton. Voici ma solution :

Contenu masqué n°4

Pour ce second défi, la réponse n'a pas dû être bien longue à trouver puisqu'elle se trouvait déjà dans votre code. Voici ma solution :

Contenu masqué n°5

Enfin, ce dernier défi a dû vous demander un peu plus de réflexion. Voici ce que je vous propose :

Contenu masqué n°6

Vous voilà capable de créer des fenêtres qui réagissent à votre utilisateur ! Mais au fait... Celle-ci ne se compose que d'un seul widget non ? C'est un peu nul.
Donc le prochain chapitre sera consacré aux layouts ! Vous allez pouvoir mettre autant de widgets que vous voudrez dans vos fenêtres grâce à eux.

Contenu masqué

Contenu masqué n°4

```
from gi.repository import Gtk


def on_button_clicked(button):
    ''' Change le label du bouton '''
    button.set_label('Pouet!')

window = Gtk.Window()

button = Gtk.Button(label='Chiche de cliquer ?')
window.add(button)

button.connect('clicked', on_button_clicked)
window.connect('delete-event', Gtk.main_quit)

window.show_all()
Gtk.main()
```

Retourner au texte.

Contenu masqué n°5

```
from gi.repository import Gtk


window = Gtk.Window()

button = Gtk.Button(label='Chiche de cliquer ?')
window.add(button)

button.connect('clicked', Gtk.main_quit)
window.connect('delete-event', Gtk.main_quit)
```

```
11
12 window.show_all()
13 Gtk.main()
```

Retourner au texte.

Contenu masqué n°6

```
1  from gi.repository import Gtk
2
3
4  def open_window(button):
5      ''' Ouvre une nouvelle fenêtre '''
6      sub_window = Gtk.Window()  # On créer une nouvelle fenêtre
7
8      close_btn = Gtk.Button(label='Fermer l\'application')
9      sub_window.add(close_btn)
10
11     # On connecte l'événement clicked du bouton au callback
           main_quit de Gtk
12     close_btn.connect('clicked', Gtk.main_quit)
13
14     sub_window.show_all()  # On affiche le tout
15
16
17 window = Gtk.Window()
18
19 open_btn = Gtk.Button(label='Chiche de cliquer ?')
20 window.add(open_btn)
21
22 open_btn.connect('clicked', open_window)
23 window.connect('delete-event', Gtk.main_quit)
24
25 window.show_all()
26 Gtk.main()
```

Retourner au texte.

4. Le positionnement grâce aux layouts

Il est temps de complexifier un peu nos fenêtres. Car un widget c'est bien mais... Ça pourrait être mieux ! Mais pour en ajouter, il va nous falloir étudier les layouts.
Le positionnement, c'est toute une science. On peut facilement se perdre dans la multitude des options proposées. C'est pour ça que l'on va utiliser un constructeur d'interface, Glade, dans quelques chapitres pour nous simplifier la vie. Mais pour le moment, nous allons tout faire à la main (et je sais que vous aimez ça !).

4.1. Introduction

La plupart des bibliothèques GUI requièrent de vous de placer précisément vos widgets dans une fenêtre, en utilisant le positionnement absolu ou relatif, GTK+ vous propose une approche du problème différente. Plutôt que de préciser la taille et la position d'un widget, vous allez les organiser en ligne ou en colonne par rapport aux autres. Vous pourrez évidemment forcer un widget de prendre la taille que vous souhaitez si vous en avez réellement le besoin, mais GTK+ n'a pas été pensé pour ça. Pour ce positionnement, nous allons utiliser des conteneurs.
De ce placement *intelligent* s'en résulte quelques propriétés :

- La taille de votre widget sera déterminée par la taille de son contenu. Une taille minimale pourra être spécifiée.
- Un widget occupera le maximum de place qu'il pourra. Il *remplira* l'espace qui lui est fournit.
- La largeur et la hauteur d'une ligne ou d'une colonne d'un conteneur sera définie à partir du plus large/haut widget qu'il contient.
- La taille de votre fenêtre sera automatiquement calculée en fonction de la taille des widgets qu'elle contient.

Ça peut paraître restrictif et compliqué comme ça, mais en fait c'est un fonctionnement vraiment logique.
Il faut aussi savoir qu'un conteneur est un widget. Ça signifie que l'on peut mettre un conteneur dans un conteneur sans aucun problème. Et c'est là que résulte toute la puissance de ce système.

Un grand pouvoir implique de grandes responsabilités. En effet, les widgets sont un système vraiment efficace. Cependant, si vous souhaitez les emboîter, votre architecture risque de devenir très complexe et vous allez vous perdre. Je vous conseil donc deux solutions :

- Dessiner sur une feuille votre application afin d'avoir un modèle
- Attendre patiemment la partie de ce tutoriel à propos de Glade

Nous allons utiliser deux types de conteneurs, les autres étant spécifiques à certaines utilisations. Le type `Gtk.Box` et le type `Gtk.Grid`.

4.2. Les boîtes

Le type `Gtk.Box` est sûrement le conteneur le plus simple qu'il soit. Il vous permet de de créer des lignes ou des colonnes de widgets. Un `Gtk.Box` équivaut à une ligne ou une colonne. Voyez l'exemple ci-dessous :

```
1  from gi.repository import Gtk
2
3
4  # On créer notre fenêtre, comme d'habitude
5  window = Gtk.Window()
6  window.set_border_width(10)  # Je met une bordure pour que ce soit
       visuellement plus joli
7  window.connect('delete-event', Gtk.main_quit)
8
9  # On créer un conteneur de type Gtk.Box()
10 box = Gtk.Box(spacing=6)  # L'argument "spacing" équivaut à
       l'écartement entre les widgets
11
12 # On créer deux boutons totalement bidons
13 button_1 = Gtk.Button(label='Je suis le bouton n°1')
14 button_2 = Gtk.Button(label='Je suis le bouton n°2')
15
16 # Et le plus important, on "pack" les boutons dans notre box
17 # c'est-à-dire qu'on les ajoute dedans
18 # Le bouton 1 :
19 box.pack_start(button_1, True, True, 0)
20 # Le bouton 2 :
21 box.pack_start(button_2, True, True, 0)
22
23 # Et enfin on ajoute ce conteneur à notre fenêtre
24 window.add(box)
25
26 window.show_all()
27 Gtk.main()
```

Voici le résultat :

FIGURE 4.1.

Ici, pas mal de nouveauté. Tout d'abord, je créer mon conteneur avec :

```
box = Gtk.Box(spacing=6)
```

`box` utilise le type par défaut de conteneur, c'est à dire qu'il sera horizontal. De plus, tous ces enfants seront espacés de 6 pixels, mais attention, juste à droite et à gauche ! Comme le conteneur est horizontal, l'espacement se fait horizontalement. L'inverse se produit quand le conteneur est vertical.
Je créer ensuite mes deux boutons en leur assignant un label, rien de nouveau pour vous (du moins, je l'espère).
L'action que je fais ensuite s'appelle *packer*. C'est à dire que j'ajoute, j'assigne mes boutons à mon conteneur. *Je pack mes boutons.* Pour ça, j'utilise la méthode `Gtk.Box.pack_start()` :

```
# Le bouton 1 :
box.pack_start(button_1, True, True, 0)
# Le bouton 2 :
box.pack_start(button_2, True, True, 0)
```

Cette méthode ajoute, de gauche à droite, vos widgets. Elle prend 4 arguments, tous obligatoires. Le premier est simplement le widget à ajouter. Un bouton, une image, un autre conteneur... Tout ce que vous voulez qui soit un widget.
Le deuxième est booléen. Il permet que si plus d'espace est disponible, d'autoriser votre conteneur à diviser cet espace en part égale pour chaque widget. Par exemple, si j'agrandis la fenêtre, plus d'espace sera disponible, donc nous verrons le widget *s'agrandir*. Si cet argument est sur False, alors cette *case* ne sera pas agrandie. Essayer d'agrandir la fenêtre avec cette valeur à `False`.
Pour les plus flemmards d'entre vous, petite démonstration avec un gif :

FIGURE 4.2. – Le bouton 1 a son redimensionnement réglé sur False tandis que celui du bouton 2 est réglé sur True

Nous travaillons dans un conteneur horizontal, cette propriété n'affecte que l'axe horizontal. Il est donc parfaitement normal que le widget voit sa hauteur modifiée.

Le troisième argument y ressemble un peu. C'est encore un booléen sauf que là l'espace disponible pour le widget va s'agrandir sans que celui-ci (le widget) ne cherche à le remplir. Pour bien comprendre, il vous faut absolument tester par vous même et tant pis pour les plus flemmards, pas de gif cette fois. Il faut, en tout logique, que le second argument soit sur vrai (pour autoriser la box à agrandir la case mais pour interdire le widget à remplir cette case).
Et enfin, le dernier argument est un entier. Il correspond à la taille des marges à gauche et à droite de ce widget. Il faut que cette taille soit supérieur à l'espacement entre les widgets pour en voir les effets.
Sachez qu'il existe la même méthode mais qui ajoute les widgets de droite à gauche, `Gtk.Box.pack_end()`.
Ici nous avions un conteneur horizontal. Je vous présente le même, mais vertical :

FIGURE 4.3.

Et voici comment faire :

```
box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
```

Par défaut c'est horizontal, mais si pour une raison que je ne connais pas, vous avez besoin de le préciser :

```
box = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=6)
```

Et voilà, vous savez tout à propos des conteneurs de type boîte. Nous allons voir le conteneur de type Grille, qui est plus simple à comprendre je trouve et beaucoup plus complet. Par contre, je le trouve moins *précis*.

4.3. Les grilles

Les grilles seront sûrement un des widgets que vous utiliserez le plus au début. En effet, elles sont plus simple à manipuler et permettent beaucoup plus de chose. Seulement, au moment ou vous commencerez à faire des fenêtre avec beaucoup de widgets vous découvrirez leur principal défaut.
Voyez les grilles comme un tableau de LibreOffice Calc ou Excel. C'est à dire des cases qui appartiennent à des lignes et des colonnes. Vous allez attacher vos widgets à ces cases et vous allez pouvoir leur indiquer de prendre 3 case de large, 6 cases de haut si vous voulez.
Voici un exemple d'utilisation des grilles :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  window = Gtk.Window()
7
8  window.set_title('Demo des grilles')
9  window.set_border_width(10)
10
11 window.connect('delete-event', Gtk.main_quit)
12
13 grid = Gtk.Grid()
14
15 button_1 = Gtk.Button(label='Je suis le bouton n°1')
16 button_2 = Gtk.Button(label='Je suis le bouton n°2')
17 button_3 = Gtk.Button(label='Je suis le bouton n°3')
18 button_4 = Gtk.Button(label='Je suis le bouton n°4')
19 button_5 = Gtk.Button(label='Je suis le bouton n°5')
20
21 grid.attach(button_1, 0, 0, 3, 1)  # Le bouton_1 se trouve en
       (0;0), prend 3 cases de large et une de haut
22 grid.attach(button_2, 0, 1, 1, 3)  # Le bouton_2 se trouve en
       (0;1), prend une case de large et 3 de haut
23 grid.attach(button_4, 1, 1, 1, 1)  # Le bouton_4 se trouve en
       (1;1), prend une case de large et une de haut
24 grid.attach(button_3, 2, 1, 1, 1)  # Le bouton_3 se trouve en
       (2;1), prend une case de large et une de haut
25 grid.attach(button_5, 1, 3, 2, 1)  # Le bouton_5 se trouve en
       (1;3), prend 2 cases de large et une de haut
26
27 window.add(grid)
28
```

```
29 window.show_all()
30 Gtk.main()
```

Et le résultat en image :

FIGURE 4.4.

Premièrement, il faut créer la grille :

```
1 grid = Gtk.Grid()
```

Rien de nouveau. Ce qui change, c'est comment vous allez attacher vos widgets à cette grille. En effet, on ne *pack* pas ses widgets, on les *attache*.

```
1 grid.attach(widget, x, y, size_x, size_y)
```

La méthode `Gtk.Grid.attach()` prend 5 arguments obligatoires. Le premier est le widget que vous souhaitez ajouter. Le second et le troisième sont les coordonnées de ce widget. Respectivement, les abscisses et les ordonnées.

FIGURE 4.5. – Coordonnées d'une Gtk.Grid

Les deux derniers , `x_size` et `y_size` représentent le nombre de case que votre widget prendra. Faite attention de ne pas mettre cette valeur à 0 sinon votre widget n'apparaîtra pas et vous aurez un avertissement :

```
1 Gtk-CRITICAL **: gtk_grid_attach: assertion 'height > 0' failed
```

De plus, faite aussi attention à ce que vous ne recouvriez pas les autres widgets. Je pense que le meilleur moyen de maîtriser ce conteneur est de pratiquer, non ?
Mais avant de vous laisser vous débrouiller, j'aimerais vous présentez quatre nouvelles méthodes.
Tout d'abord, `Gtk.Grid.set_column_homogeneous()` et `Gtk.Grid.set_row_homogeneous()`. Si vous avez essayé de faire une grille, vous avez peut-être remarqué que quand vous agrandissiez la fenêtre, la grille (et donc les widgets contenus dedans) ne suivait pas. En effet, votre grille n'est, par défaut, pas *homogène*.

`Gtk.Grid.set_column_homogeneous()` prend en argument un booléen et permet d'autoriser ou non les colonnes de la grille à "suivre" le mouvement de la fenêtre. Pour simplifier, ça signifie que vos widgets contenus dans cette grille vont s'agrandir horizontalement.
`Gtk.Grid.set_row_homogeneous()`, comme sa sœur, prend en argument un booléen. Sauf qu'ici, ce sont les lignes qui vont avoir le droit de s'agrandir ou non.
Les deux derniers vont aussi ensembles. Grâce à `Gtk.Grid.set_row_spacing()` et Gtk.Grid.set_column_spacing()' vous allez pouvoir définir l'espace entre chacun de vos widgets verticalement ou horizontalement. Je vous laisse jouer avec pour en découvrir les effets.

4.4. Un compteur de clics

Allez on est parti !
Je voudrais (et oui, j'exige maintenant) que vous me fassiez une fenêtre qui compte le nombre de clics sur un bouton. Voici ce que j'attends :

FIGURE 4.6. – Résultat de l'exercice, une fenêtre qui compte le nombre de clique !

Si vous réussissez ce défi, c'est que vous avez tout compris jusque là. Si vous n'y arrivez pas, ne vous précipitez pas sur la solution, allez relire les chapitres où vous bloquez. Et si ce qui vous pose problème est ce compteur et non l'interface, voici un indice :

> Contenu masqué n°7

Ma solution (parce que vous avez sûrement trouvé une autre façon de faire !) :

> Contenu masqué n°8

Ceux qui, comme moi, on utilisé une grille on le droit à un bouton *Quitter* énorme. En effet, le label au-dessus étant plus large, la colonne s'agrandit en conséquence. Et le bouton quitter également. Plusieurs solutions pour résoudre ce problème :

- Interdire le bouton de remplir l'espace en réglant son alignement horizontal sur centré (par défaut il remplit tout l'espace) avec `button.set_halign(Gtk.Align.CENTER)`
- Utiliser des `Gtk.Box` (solution que je préfère)
- Déplacer le bouton

Autre problème, c'est cette variable globale. Ce problème sera résolu grâce à la POO que nous verrons plus tard.

Vous avez désormais vu le plus gros de GTK+. Dans le prochain chapitre, je vais vous présenter quelques widgets que je trouve utile ou important. Il ne vous manque plus que la POO pour pouvoir commencer à faire des *vraies* fenêtres !

Contenu masqué

Contenu masqué n°7

```
global mon_compteur
```

Retourner au texte.

Contenu masqué n°8

```
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk


def build_interface():
    '''
    Fonction qui retourne le layout contenant toute notre interface
    '''
    grid = Gtk.Grid()

    grid.set_row_spacing(6)
    grid.set_column_spacing(6)

    click_counter_label =
        Gtk.Label('Vous n\'avez pas encore cliqué sur le bouton.')
    grid.attach(click_counter_label, 1, 0, 1, 1)

    add_counter_btn = Gtk.Button(label='Cliquez ici !')
    add_counter_btn.connect('clicked', on_add_counter,
        click_counter_label)
    grid.attach(add_counter_btn, 0, 0, 1, 1)

    quit_btn = Gtk.Button(label='Quitter')
    quit_btn.connect('clicked', Gtk.main_quit)
    grid.attach(quit_btn, 1, 1, 1, 1)

    return grid


def on_add_counter(button, label):
    '''
    Quand l'utilisateur clique sur le bouton, ajoute +1 au compteur et change le
    label
    '''
    global click_counter  # On récupère le compteur
```

```
36      click_counter += 1
37      # On change le label avec le nouevau texte
38
            label.set_text('Vous avez cliqué {} fois sur le boutton !'.format(click_
39
40
41  if __name__ == '__main__':
42      window = Gtk.Window()
43
44      window.set_title('Exercice n°1')
45      window.set_border_width(10)
46      window.connect('delete-event', Gtk.main_quit)
47
48      click_counter = 0  # Notre compteur de clics
49
50      # On récupère le layout et on l'ajoute à la fenêtre
51      grid = build_interface()
52      window.add(grid)
53
54      window.show_all()
55      Gtk.main()
```

Retourner au texte.

5. À la découverte de nouveaux widgets !

Vous connaissez pour le moment 5 widgets :

- `Gtk.Window`
- `Gtk.Button`
- `Gtk.Label`
- `Gtk.Box`
- `Gtk.Grid`

Or, GTK+ 3 est ne se limite pas à eux ! Je vais donc compléter cette petite liste avec 5 nouveaux types de widget qui sont pour moi les plus importants.

5.1. Les propriétés

Les propriétés décrivent la configuration et l'état d'un widget. Comme pour les signaux, chaque widget à ses propres propriétés. C'est pour ça qu'il faut toujours garder la documentation sous la main. Vous pouvez spécifier ces propriétés lors de la création du widget ou après, c'est selon vos préférences. Par exemple :

```
1 button = Gtk.Button(label='Un label')
2 # ou alors
3 button = Gtk.Button()
4 button.set_label('Un label')
```

Ces deux façon de faire donneront le même résultat. Il existe encore une dernière possibilité de construire un bouton même si je ne lui trouve pas d'intérêt :

```
1 button = Gtk.Button.new_with_label('Un label')
```

? Et si je veux modifier ou avoir accès aux valeurs des propriétés ?

La première solution est celle que j'utilise la plupart du temps :

```
1 button.set_label('Un nouveau label')  # Défini un nouveau label
2 label_button = button.get_label()  # Retourne le label du boutton
```

Une seconde solution :

```
1 button.set_property('label', 'Un nouveau label')
2 label_button = button.get_property('label')
```

Enfin, il arrivera peut être qu'un jour vous ayez besoin d'afficher les différentes propriétés d'un widget :

```
>>> button = Gtk.Button(label='Salut')
>>> print(dir(button.props))
['action_name', 'action_target', 'always_show_image',
    'app_paintable', 'border_width', 'can_default', 'can_focus',
    'child', 'composite_child', 'double_buffered', 'events',
    'expand', 'focus_on_click', 'halign', 'has_default',
    'has_focus', 'has_tooltip', 'height_request', 'hexpand',
    'hexpand_set', 'image', 'image_position', 'is_focus', 'label',
    'margin', 'margin_bottom', 'margin_end', 'margin_left',
    'margin_right', 'margin_start', 'margin_top', 'name',
    'no_show_all', 'opacity', 'parent', 'receives_default',
    'related_action', 'relief', 'resize_mode', 'scale_factor',
    'sensitive', 'style', 'tooltip_markup', 'tooltip_text',
    'use_action_appearance', 'use_stock', 'use_underline',
    'valign', 'vexpand', 'vexpand_set', 'visible', 'width_request',
    'window', 'xalign', 'yalign']
```

5.2. Les labels

Si vous voulez afficher du texte, vous le ferez grâce aux labels. Ils vous offrent un nombre d'option impressionnant.

Je le rappelle encore une fois, mais gardez votre documentation ouverte à la page du widget en question. Pour les labels, c'est par ici .

Un label se construit de cette façon :

```
label = Gtk.Label('Je suis un label')
# Ou alors
label = Gtk.Label()
label.set_text('Je suis un label')
```

Vous pouvez rendre ce label sélectionnable avec `Gtk.Label.set_selectable()` qui prend en argument un booléen. Ça signifie que l'utilisateur pourra sélectionner votre texte et le copier avec un clique droit par exemple. Il ne pourra pas le modifier.
GTK+ impose quelques règles afin que les développeurs ne fassent pas n'importe quoi et qu'il y ai une unicité entre les différents programmes. Seuls les labels contenant une information utile — comme les messages d'erreurs — devraient être fait sélectionnable.

C'est tristounet... Si je veux lui donner du style ?

Vous voulez le pimper ? Pimpons-le alors !
On commence par l'alignement du texte. Par défaut, il est aligné à gauche. Voici les différentes

justification possible :

```
label.set_justify(Gtk.Justification.LEFT)  # Justifié à gauche
label.set_justify(Gtk.Justification.CENTER)  # Justifié au centre
label.set_justify(Gtk.Justification.RIGHT)  # Justifié à droite
label.set_justify(Gtk.Justification.FILL)  # Le texte est réparti
    dans le label pour le remplir
```

Pour changer la couleur, la taille, la police, nous allons utiliser des balises. GTK utilise Pango pour faire le rendu. Pango est également capable d'afficher tous les caractères spéciaux que vous souhaitez.
Pour indiquer à GTK que vous voulez utiliser Pango, définissez le texte de votre label grâce à la méthode `Gtk.Label.set_markup()` et non plus `Gtk.Label.set_text()`.
Les différentes balises disponibles :

Balises	Effet
	Met le texte en gras
	Met le texte en italique
<u></u>	Souligne le texte
<s></s>	Barre le texte
<big></big>	Incrémente la taille du texte
	Décrémente la taille du texte
<tt></tt>	Met le texte en télétype
	Met le texte en exposant
	Met le texte en indice

Pour ceux qui ne savent pas comment ça s'utilise :

```
<b>Je suis un texte en gras !</b>
<i>Et moi en italique !</i>
Bah moi je suis rien... :'(
```

Vous l'aurez compris, il suffit de mettre votre texte entre une balise ouvrante `<>` et une balise fermante `</>`.
Bien maintenant vous voulez changer la couleur ou la police ? On va utiliser la balise `<span></span>` qui prend tous les attributs de votre texte. Par exemple :

```
label = Gtk.Label()
label.set_markup('<span color="#c0392b" weight="bold" font="FreeSerif">Je suis
```

FIGURE 5.1. – Un magnifique texte - Presque de l'art

Il faut évidemment que la police soit présente sur l'ordinateur !
Voici les différents attributs :

Attributs	Effet
face/font/font_family	Définit la police d'écriture. Veillez à verifier que celle-ci soit installé sur l'ordinateur de votre utilisateur.
size	La taille de la police en pixels. Peut aussi être *xx-small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*
style	Définit le style des caractères : *normal*, *oblique* ou *italic*
weight	Définit l'épaisseur des caratcères : *ultralight*, *light*, *normal*, *bold*, *ultrabold*, *heavy* ou une valeur numérique
variant	Par défaut sur la valeur *normal*, permet de mettre le texte en petite majuscules avec *smallcaps*
stretch	Définit l'espacement entre les caratcères : *ultracondensed*, *extracondensed*, *condensed*, *semicondensed*, *normal*, *semiexpanded*, *expanded*, *extraexpanded* ou *ultraexpanded*.
foreground	Définit la couleur du texte. La valeur doit être en hexadécimale (#000000 par exemple)
background	Définit la couleur d'arrière plan du texte. La valeur doit être en hexadécimale (#000000 par exemple)
underline	Définit le soulignement du texte : *single*, *double*, *low* ou *none*
underline_color	Définit la couleur du soulignement. La valeur doit être en hexadécimale (#000000 par exemple)
rise	Définit l'élévation du texte (en indice ou exposant) en valeur décimale. Les valeurs négatives sont possibles pour mettre le texte en indice
strikethrough	Si True, barre le texte. Vaut False par défaut
strikethrough_color	Définit la couleur de la rature. La valeur doit être en hexadécimale (#000000 par exemple)
lang	Définit la langue du texte

Comme vous pouvez le voir, il y en a pas mal. Vous avez de quoi vous amuser avec tout ça !
Petit bonus, je vais vous montrer comment faire des infobulles. Elles apparaissent quand vous laissez votre curseur sur un widget pendant quelques secondes. Elle vous informera de l'effet du widget auquel elle appartient.
Cette fonctionnalité appartenant à `Gtk.Widget`, toutes les classes héritant possède donc cette fonctionnalité, c'est à dire tous les widgets de GTK+. Voici comment ça s'utilise :

```
widget.set_tooltip_text('Ma super infobulle')  # Avec un simple
    texte
widget.set_tooltip_markup('<i>En italique</i>')  # Vous l'aurez
    compris, vous pouvez également utiliser Pango dans les
    infobulles !
```

5.3. Les champs de texte

Un autre widget super important : le `Gtk.Entry`. J'aborderais également le `Gtk.SpinButton` ici.
Ces widgets permettent à l'utilisateur d'entrer du texte. Par exemple si vous avez un formulaire, vous allez demander le nom et le prénom de votre utilisateur à travers un `Gtk.Entry`. Voici comment il s'utilise :

```
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk


def on_entry_activated(entry, label):
    '''
    Est appellée quand entry est activé, récupère son texte et l'affiche dans
    label, puis nettoie l'entry
    '''
    text = entry.get_text()  # On récupère le texte
    label.set_text(text)  # On l'affiche dans label
    entry.set_text('')  # On vide l'entry


window = Gtk.Window()
window.set_title('Perroquet')
window.set_border_width(10)
window.connect('delete-event', Gtk.main_quit)

layout = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)

parrot = Gtk.Label('Entrez votre texte & appuyez sur entrée...')

entry = Gtk.Entry()
entry.connect('activate', on_entry_activated, parrot)  # Quand
    l'utilisateur appuie sur <Entrée>

layout.pack_start(entry, True, True, 0)
layout.pack_start(parrot, True, True, 0)

```

```
32 window.add(layout)
33 window.show_all()
34 Gtk.main()
```

Le résultat :

FIGURE 5.2. – *Bravo à ceux qui trouverons la référence au film d'animation !*

Vous l'aurez compris, `Gtk.Entry.get_text()` récupère le texte (toujours de type string) et `Gtk.Entry.set_text(str)` définit le texte. Il existe pas mal d'options, notamment mettre un placeholder avec `Gtk.Entry.set_placeholder_text(str)`.
Je vous ai également parlé d'un `Gtk.SpinButton`. C'est un widget qui hérite de `Gtk.Entry` et qui offre deux boutons en plus, un pour incrémenter et un autre pour décrémenter. Si vous avez des valeurs numériques à demander à votre utilisateur, c'est le meilleur moyen. Voici la tête que ça a :

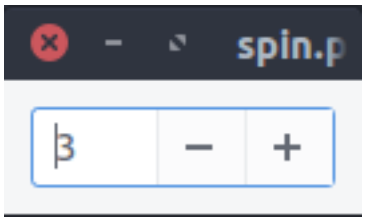

FIGURE 5.3. – Un spinbutton de Gtk+

Voici un code implémentant un *SpinButton* :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  window = Gtk.Window()
7  window.set_border_width(10)
8  window.connect('delete-event', Gtk.main_quit)
9
10 adjustment = Gtk.Adjustment(0, 0, 100, 1, 10, 0)
11 spin_button = Gtk.SpinButton()
12 spin_button.set_adjustment(adjustment)
13
14 window.add(spin_button)
15 window.show_all()
16 Gtk.main()
```

Comme vous le voyez, on a un nouveau widget qui vient faire son trouble-fête, le `Gtk.Adjustment`. En effet, votre `Gtk.SpinButton` demande qu'on lui fournisse ce nouvel élément afin de savoir :

— sa valeur actuelle
— sa valeur minimale
— sa valeur maximale
— et son pas, c'est à dire de combien en combien il doit changer sa valeur (de 1 en 1, de 5 en 5...)

C'est exactement les arguments que j'ai mis quand j'ai initialisé le `Gtk.Adjustement`. Les deux derniers ne nous intéressent pas pour le moment, ils vont servir au moment où l'on va s'intéresser au scroll des pages. Mettez `10` et `0` et tout se passera très bien.
La valeur de cette entrée se récupère avec `Gtk.SpinButton.get_value()`, elle sera de type `float`. Tout naturellement, pour assigner une valeur c'est `Gtk.SpinButton.set_value()`.

5.4. Les toggleButtons et les switchs

Comme ils fonctionnent un peu de la même façon, j'ai décider de tout mettre ensemble. Ici, nous allons aborder les toggleButtons et les switchs. Commençons par le premier.
Les toggleButtons vous avez dû en voir souvent, rien que sur ce site. Ce sont en effet des petites cases que l'on peut cocher ou décocher. Ils existent aussi sous une autre forme, celle des boutons. Il y aussi les boutons de type radio, eux, vous ne pouvez en sélectionner qu'un seul parmi les autres. Pour être plus clair, voici un code que je vous recommande fortement d'essayer chez vous :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  window = Gtk.Window()
7  window.set_title('Des boutons partout !')
8  window.set_border_width(10)
9  window.connect('delete-event', Gtk.main_quit)
10
11 main_layout = Gtk.Grid()
12 main_layout.set_column_spacing(6)
13 main_layout.set_row_spacing(6)
14 main_layout.set_row_homogeneous(True)
15
16 #======================
17 # Bouton de type radio
18 #======================
19
20 # On créer un premier radio avec un label
21 radio_1 = Gtk.RadioButton.new_with_label(None, 'Pilule bleu')
22
23 # On en créer un second en lui indiquant qu'il appartient au même
       groupe que le radio_1
```

```
24 radio_2 = Gtk.RadioButton.new_from_widget(radio_1)
25 radio_2.set_label('Pilule rouge')  # On lui met un label
26
27 # On fait tout en même temps
28 radio_3 = Gtk.RadioButton.new_with_label_from_widget(radio_1,
       'Je n\'ai pas compris la référence...')
29
30 main_layout.attach(radio_1, 0, 0, 1, 1)
31 main_layout.attach(radio_2, 1, 0, 1, 1)
32 main_layout.attach(radio_3, 2, 0, 1, 1)
33
34 #==================
35 # Bouton persistant
36 #==================
37
38 # Voyez ça comme des interupteurs
39 toggle_1 = Gtk.ToggleButton('Faire un café')
40 toggle_2 = Gtk.ToggleButton('Acheter un croissant')
41 toggle_3 = Gtk.ToggleButton('Dormir')
42
43 main_layout.attach(toggle_1, 0, 1, 1, 1)
44 main_layout.attach(toggle_2, 1, 1, 1, 1)
45 main_layout.attach(toggle_3, 2, 1, 1, 1)
46
47 #===============
48 # Cases à cocher
49 #===============
50
51 # Ils fonctionnent exactement de la même façon que les boutons
       persistants sauf
52 # qu'ils rajoutent une petite case à cocher
53
54 check_1 = Gtk.CheckButton('Faire un café')
55 check_2 = Gtk.CheckButton('Acheter un croissant')
56 check_3 = Gtk.CheckButton('Dormir')
57
58 main_layout.attach(check_1, 0, 2, 1, 1)
59 main_layout.attach(check_2, 0, 3, 1, 1)
60 main_layout.attach(check_3, 0, 4, 1, 1)
61
62 window.add(main_layout)
63 window.show_all()
64 Gtk.main()
```

Le résultat :

http://img11.hostingpics.net/pics/972724Bouton

FIGURE 5.4.

Petite explication pour les boutons de type radio. Comme je vous l'ai expliqué, ils fonctionnent par groupe. J'ai créé le premier avec `Gtk.RadioButton.new_with_label(None, 'Pilule bleu')`. Ce constructeur prend en premier argument un tableau contenant au moins un membre du groupe. Si vous souhaitez créer un nouveau groupe, rentrez `None`. J'utilise également `Gtk.RadioButton.new_from_widget(radio_1)` qui prend comme seul argument un membre du groupe (mais pas un tableau cette fois-ci) ou `None` pour créer un nouveau groupe. Et enfin, la méthode `Gtk.RadioButton.new_with_label_from_widget(radio_1, 'Je n\'ai pas compris la référence...')` permet de tout faire d'un coup. Elle prend en premier argument un membre du groupe ou `None` pour créer un nouveau groupe puis son label.
Pour récupérer la valeur d'un de ces boutons, il vous suffit de faire un `Gtk.ToggleButton.toggled()` qui vous retournera un booléen. `True` si le bouton est coché/activé et `False` sinon.
Passons aux switchs. Encore un bouton, mais que ne prend que deux états : *On* ou *Off*.

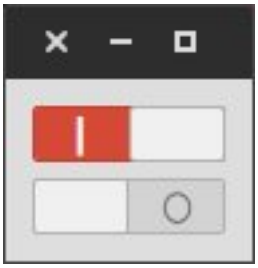

FIGURE 5.5. – Démonstration des switchs

Le code de cette minuscule fenêtre :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  window = Gtk.Window()
7  window.set_title('Switch me !')
8  window.set_border_width(10)
9  window.connect('delete-event', Gtk.main_quit)
10
11 main_layout = Gtk.Box(orientation=Gtk.Orientation.VERTICAL,
       spacing=6)
12
13 switch_on = Gtk.Switch()
14 switch_on.set_active(True)  # Je force le premier switch à passer
       sur "On"
15
16 switch_off = Gtk.Switch()  # Le second est par défaut sur "Off"
17
18 main_layout.pack_start(switch_on, True, True, 0)
19 main_layout.pack_start(switch_off, True, True, 0)
20
21 window.add(main_layout)
```

```
22 window.show_all()
23 Gtk.main()
```

Pour récupérer la valeur d'un switch, faite simplement un `Gtk.Switch.get_active()` qui vous retournera soit `True` soit `False`.

5.5. Les images avec Pixbuf

Pour toutes les images que vous voudriez mettre dans votre GUI, il faudra passer par un Pixbuf. Je vous rassure, rien de douloureux. Voici le lien de la documentation de `GdkPixbuf` : http ://lazka.github.io/pgi-docs/GdkPixbuf-2.0/ .
Il existe deux façons de mettre des images dans votre interface. Soit, comme dit dans l'introduction, de passer par un Pixbuf, soit de l'importer directement dans le `Gtk.Image`. GTK+ conseille la première méthode, je vous la recommande aussi.
Avec un Pixbuf :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk, GdkPixbuf
5
6  window = Gtk.Window()
7  window.set_title('Wow une image')
8  window.connect('delete-event', Gtk.main_quit)
9
10 # On créer notre Pixbuf
11 image = GdkPixbuf.Pixbuf.new_from_file('patrick.png')
12 image_renderer = Gtk.Image.new_from_pixbuf(image)
13
14 window.add(image_renderer)
15 window.show_all()
16
17 Gtk.main()
```

N'oublier pas d'importer `GdkPixbuf` ! Voici le superbe rendu :

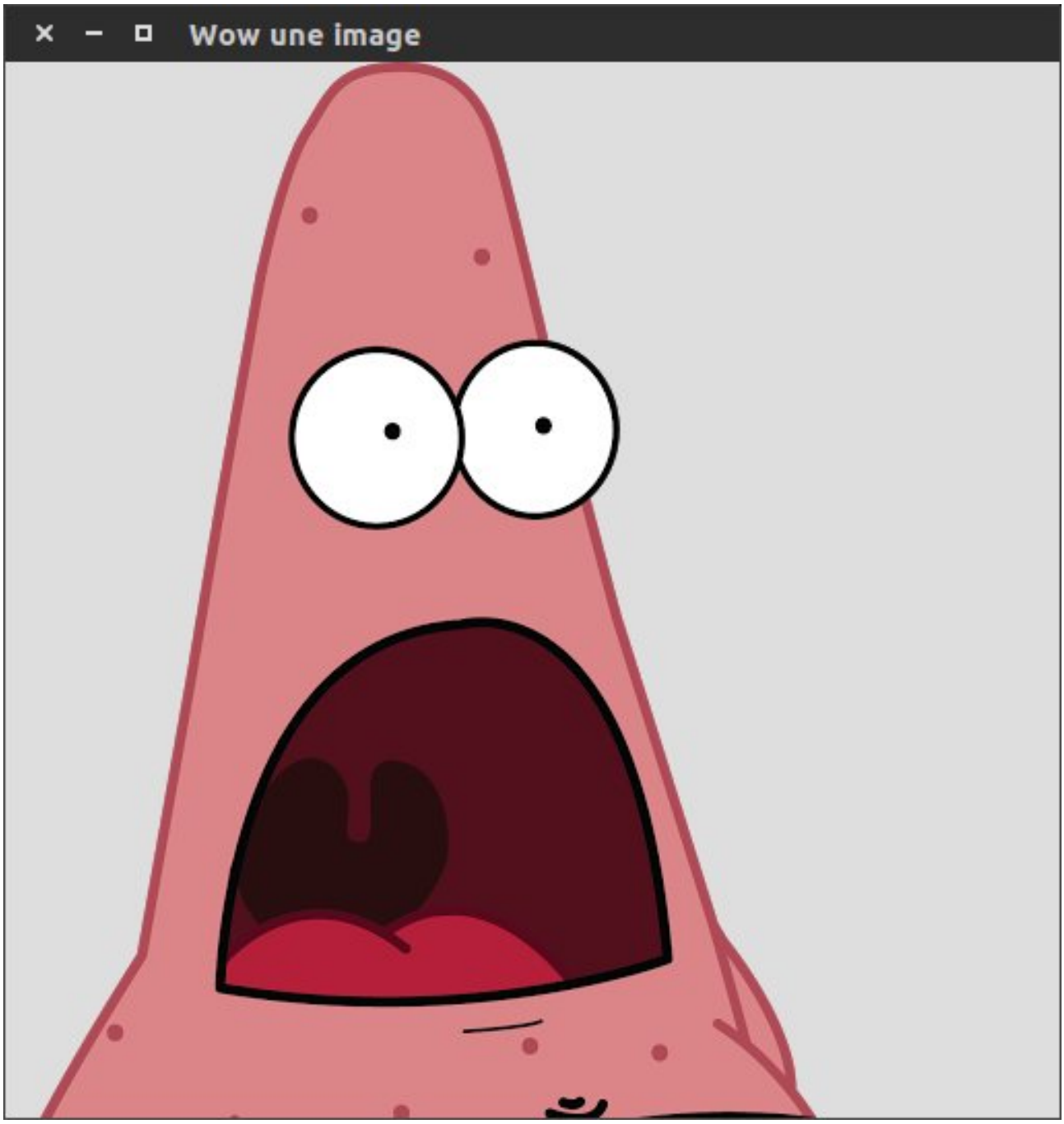

FIGURE 5.6. – *So much wow !*

Le même rendu sera obtenu avec ce code (du coup, sans le Pixbuf) :

```
#!/usr/bin/env python3
# coding: utf-8

from gi.repository import Gtk

window = Gtk.Window()
window.set_title('Wow une image')
window.connect('delete-event', Gtk.main_quit)
```

```
 9
10 image_renderer = Gtk.Image.new_from_file('patrick.png')
11
12 window.add(image_renderer)
13 window.show_all()
14
15 Gtk.main()
```

L'intérêt des Pixbuf c'est que vous pouvez modifier votre image directement depuis votre fenêtre (incliner l'image, la redimensionner, etc). Je vous laisse chercher dans la documentation pour savoir comment faire.

N'oubliez pas que plus une image est lourde, plus elle met du temps à charger. Si vous trouvez que votre application est lente et que vous utilisez des Pixbuf, essayez de compresser vos images.

Autre intérêt, ce sont les GIFs . Vous avez pu en voir déjà pas mal tout au long de ce tutoriel et vous continuerez à en voir car j'adore ça ! Je parle bien sûr de ces images animées. Comment en mettre dans vos logiciels ? Pas le choix ici, vous êtes obligé de passer par un `Gdk.Pixbuf`, mais pas n'importe quel `Gdk.Pixbuf` non, un `Gdk.PixbufAnimation`.

```
 1 #!/usr/bin/env python3
 2 # coding: utf-8
 3
 4 from gi.repository import Gtk, GdkPixbuf
 5
 6 window = Gtk.Window()
 7 window.set_title('Wow une image')
 8 window.connect('delete-event', Gtk.main_quit)
 9
10 # On créer notre PixbufAnimation
11 image =
       GdkPixbuf.PixbufAnimation.new_from_file('mon_super_gif.gif')
12 image_renderer = Gtk.Image.new_from_animation(image)  # ATTENTION
       on change de méthode!
13
14 window.add(image_renderer)
15 window.show_all()
16
17 Gtk.main()
```

5.6. Les calendriers

Le calendrier a été l'un des widgets qui m'a le plus impressionné par sa simplicité. Je ne pensais pas qu'il serait aussi facile de mettre un calendrier dans un logiciel. Voyons ça ensemble, j'espère que vous serez aussi impressionné que moi !
Bon, tout le monde le sait, un calendrier c'est très complexe et sûrement pas à la porté de tout

le monde. Encore une fois, GTK+ est là et nous propose un calendrier tout fait et très complet. Regardez un peu ça :

```
1  from gi.repository import Gtk
2
3  window = Gtk.Window()
4  window.set_title('Un calendrier')
5  window.set_border_width(10)
6  window.connect('delete-event', Gtk.main_quit)
7
8  calendar = Gtk.Calendar()
9
10 window.add(calendar)
11 window.show_all()
12 Gtk.main()
```

Donnera la figure suivante.

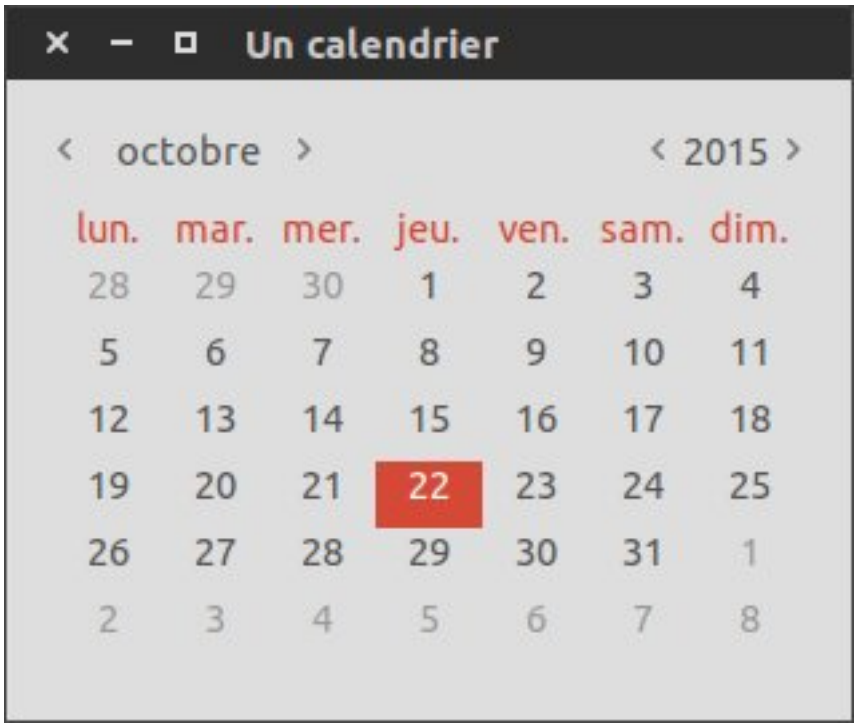

FIGURE 5.7. – Un calendrier sous GTK

Par défaut, il sera toujours réglé sur le jour actuel.
Pour changer le jour, ça se fera avec `Gtk.Calendar.select_day()` et pour changer le mois et l'année, ce sera avec `Gtk.Calendar.select_month()`. Oui je sais ça ne devrait être que les mois, mais manque de bol, ça fait les deux en même temps. D'ailleurs en parlant de mois, il faut que je vous prévienne que les indices sont déplacées d'un cran. C'est à dire que pour GTK+, juin est le cinquième mois et non le sixième, comme décembre est le onzième et non le douzième. À part ça, tout se passe très bien !
Par exemple, mettons la date au premier janvier 2000.

```
calendar.select_day(1)
calendar.select_month(0, 2000)
```

Trop facile, non ?
Pour réagir quand l'utilisateur change la date, ce sera avec le signal `Gtk.Calendar.signals.day_selected`. Petite démonstration :

```
calendar.connect('day-selected', on_date_changed)

def on_date_changed(calendar):
    year, month, day = calendar.get_date()
    month += 1  # On décale les indices
    print('Vous avez sélectionné le {}/{}/{}'.format(day, month,
        year))
```

Pour récupérer la date, c'est donc avec `Gtk.Calendar.get_date()` qui vous renvoie 3 entiers représentant dans l'ordre :

— l'année
— le mois (attention aux indices !)
— le jour

Enfin, on peut également ajouter un marqueur sur un jour avec `Gtk.Calendar.mark_day()` qui prend un entier représentant le jour. Attention, si vous le placez sur le 5 novembre, il sera placé sur tous les 5 du calendrier.

C'est bon, vous vous êtes bien reposé ? On ne va pas se le cacher, cette partie était beaucoup trop simple !
La prochaine fois on corse un peu le tout en ajoutant de la POO dans tout ça. Impatient non ?

6. [TP] Le jeu du plus ou moins

Vous connaissez certainement le plus ou moins. Vous avez peut être même une petite version qui traîne sur votre ordinateur !
Nous allons mettre ce jeu un peu à jour en le refaisant mais avec une interface graphique. Je vais donc vous faire réaliser la version minimale du jeu, ce sera à vous d'ajouter toutes les options que vous voudrez. Le but ? Obtenir un plus ou moins le plus complet au monde !

6.1. Consignes

Bon, les règles du plus ou moins sont très simples. L'ordinateur tire un nombre compris entre 0 et 100 et l'utilisateur doit le trouver. Le programme lui indiquant seulement si le nombre cherché est supérieur ou inférieur que le nombre entré.
Voici la version que je vous conseille de commencer à faire :

FIGURE 6.1. – La version la plus simple possible du jeu du plus ou moins

Ça ne se voit pas, mais en appuyant sur la touche Entrée, ça fonctionne aussi. De plus, si vous comptez utiliser vous aussi un `Gtk.SpinButton` pensez à bien désactiver l'incrémentation par les boutons + et -! Sinon l'utilisateur a juste à rester appuyé jusqu'à voir la réponse apparaître.

6.2. Correction

Et hop, voici la correction de la version basique.

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5  from random import randint
6
7
8  def on_validate(widget, label, entry=None):
9      '''
10     Fonction callback appellée quand l'utilisateur valide un nombre
11     '''
```

```
12      global random_nb
13
14      # Soit c'est un bouton qui est passé dans l'agument widget,
            soit le SpinButton
15      # On vérifie :
16      if entry is not None:
17          user_nb = int(entry.get_value())
18      else:
19          user_nb = int(widget.get_value())
20
21      # On change le texte
22      if user_nb > random_nb:
23
                label.set_text('La solution est inférieur à {}.'.format(user_nb))
24      elif user_nb < random_nb:
25
                label.set_text('La solution est supérieur à {}.'.format(user_nb))
26      else:
27
                label.set_text('Bravo, vous avez trouvé la bonne réponse !')
28
29
30  def build_interface():
31      '''
32      Construit toute l'interface et en retourne le layout la contenant
33      '''
34      # Le layout principal
35      main_layout = Gtk.Grid()
36
37      # Espacement entre les colonnes et entre les lignes
38      main_layout.set_column_spacing(6)
39      main_layout.set_row_spacing(6)
40      # On autorise l'homogénéité des lignes
41      main_layout.set_column_homogeneous(True)
42
43      # La zone de texte où l'utilisateur va pouvoir entrer un nombre
44      # Notre adjustement va de 0 à 100 en empechant l'incrémentation
45      adjustment = Gtk.Adjustment(0, 0, 100, 0, 10, 0)
46
47      # Le label qui affichera l'état de la partie
48      printer = Gtk.Label('Rentrez un nombre et validez !')
49
50      number_entry = Gtk.SpinButton()
51      number_entry.set_adjustment(adjustment)  # On lui assigne
            l'ajustement
52      number_entry.set_numeric(True)  # On accepte seulement les
            nombres
53      # L'entrée sera passée automatiquement en paramètre pas besoin
            de le préciser
54      number_entry.connect('value-changed', on_validate, printer)
```

```
55
56      # Le bouton valider
57      ok_btn = Gtk.Button(label='Valider')
58      # Ici l'entrée n'est pas passée automatiquement, on le fait
            manuellement
59      ok_btn.connect('clicked', on_validate, printer, number_entry)
60
61      # On les attache tous
62      main_layout.attach(number_entry, 0, 0, 1, 1)
63      main_layout.attach(ok_btn, 1, 0, 1, 1)
64      main_layout.attach(printer, 0, 1, 2, 1)
65
66      # On retourne le layout principal contenant toute notre
            interface
67      return main_layout
68
69  if __name__ == '__main__':
70      window = Gtk.Window()
71
72      window.set_title('Plus ou moins')  # On assigne un titre à
            notre fenêtre
73      window.set_border_width(10)  # Des bordures de 10px pour
            l'esthétisme
74
75      window.connect('delete-event', Gtk.main_quit)
76
77      main_layout = build_interface()
78      window.add(main_layout)
79
80      # On tire un nombre aléatoire entre 0 et 100 compris
81      random_nb = randint(0, 100)
82
83      window.show_all()
84      Gtk.main()
```

6.3. Améliorations

Ce programme est beaucoup trop simple ! Voici quelques idées d'amélioration :

- Un bouton pour recommencer une partie
- Sélecteur de niveau, de 0 à 100 étant le niveau bac à sable !
- Une icône pour le programme
- Un système de score
- Un classement des meilleurs scores
- Une sous fenêtre expliquant les règles
- Faire la version la plus horrible du siècle

Les possibilités sont infinies. Je suis sûr que vous allez réussir à faire quelque chose de super !

J'espère que vous vous serez bien amusé avec ce plus ou moins. Vous pouvez passer désormais

passer à la seconde partie.

Vous en savez désormais un peu plus à propos de GTK+ 3. Mais vous vous voyez mal faire des fenêtres complexes. Et c'est normal, il vous manque encore la POO et l'utilisation de Glade, un constructeur d'interface qui deviendra votre meilleur ami.

Deuxième partie

Utilisation avancée

Tout de suite, ça rigole beaucoup moins. Je vais vous montrer ici comment sont faites les fenêtres que vous pouvez voir dans les logiciels que vous utilisez tous les jours. Les développeurs ne s'embêtent pas à tout faire à la main, non. Ils utilisent la POO pour simplifier le code mais également des logiciels externes tel que des constructeurs d'interface pour ne pas à avoir faire ça manuellement.
Vous allez le voir, cette partie va franchement s'avérer très intéressante.

7. Utilisons la POO !

Enfin, la voilà, la grande POO. Je vous en parle depuis le début de ce cours, je pense qu'il serait enfin temps de s'y coller, non ?
Les exercices vont enfin pouvoir devenir un peu plus intéressant !

7.1. Comment ça se présente ?

Nous allons utiliser l'héritage ici. J'espère que vous êtes au point sur ce concept très puissant. Imaginons que vous voulez créer un logiciel. Il vous faudra une fenêtre, n'est-ce pas ? Nous allons donc créer une classe pour cette fenêtre qui héritera de `Gtk.Window`. Nous aurons ainsi accès à toutes ses propriétés, méthodes et signaux. Mais là où ça devient intéressant, c'est que l'on va pouvoir rajouter les notre ! Vous commencez à imaginer les possibilités ? Toujours pas !? Peut-être qu'une petite démonstration vous aiderait.
J'ai donc repris notre *hello world*.

```
1  from gi.repository import Gtk
2
3
4  class HelloWorld(Gtk.Window):
5
6      def __init__(self):
7          '''
8          Le constructeur de notre classe HelloWorld
9          '''
10         # On utilise le constructeur de Gtk.Window
11         Gtk.Window.__init__(self, title='Hello world!')
12
13         # On créer notre label, rien de nouveau ici
14         label = Gtk.Label('Hello world!')
15
16         # On l'ajoute à notre fenêtre, cette fenêtre
17         self.add(label)
18
19
20 # On créer notre application
21 app = HelloWorld()
22 app.show_all()
23
24 Gtk.main()
```

Certes vous ne voyez peut-être toujours pas l'intérêt de la POO ici, à part que je trouve que le code est bien plus clair. Mais si je fais ça :

```
from gi.repository import Gtk


class HelloWorld(Gtk.Window):

    def __init__(self):
        '''
        Le constructeur de notre classe HelloWorld
        '''
        # On utilise le constructeur de Gtk.Window
        Gtk.Window.__init__(self, title='Hello world!')
        self.connect('delete-event', Gtk.main_quit)
        self.set_border_width(10)

        layout = Gtk.Grid()
        layout.set_column_spacing(6)
        layout.set_row_spacing(6)

        # On fait un attribut pour y avoir accès partout
        self.clics_counter = 0

        self.label = Gtk.Label('Comment t\'appelles-tu ?')

        self.entry = Gtk.Entry()
        self.entry.connect('activate', self.on_validate)  # Si
            l'utilisateur appuie sur <Entrée>

        self.button = Gtk.Button(label='Valider')
        self.button.connect('clicked', self.on_validate)

        layout.attach(self.label, 0, 0, 1, 1)
        layout.attach(self.entry, 1, 0, 1, 1)
        layout.attach(self.button, 1, 1, 1, 1)

        self.add(layout)
        self.show_all()


    def on_validate(self, widget):
        '''
        Quand le champ de texte est validée
        '''
        # On fait tout d'un coup, on ne perd pas de temps ici ! :p

            self.label.set_text('Salut {} !'.format(self.entry.get_text()))
        self.entry.set_text('')

        # Pas besoin d'une variable globale à déclarer
        self.clics_counter += 1
```

```
48          # On accorde correctement le mot "clic"
49          if self.clics_counter > 1:
50
                    self.button.set_label('Vous en êtes à {} clics'.format(self.cl
51          else:
52
                    self.button.set_label('Vous en êtes à {} clic'.format(self.clic
53
54
55  # On créer notre application
56  app = HelloWorld()
57  Gtk.main()
```

Si vous en avez l'envie, faite la même chose sans POO, vous verrez que le code est beaucoup moins lisible. De plus, imaginez que vous faîtes un programme avec plein de fenêtres : vous allez pouvoir faire une classe pour chaque fenêtre. Par la suite vous aurez juste à faire interagir ces classes. Regardez comment c'est simple de créer une fenêtre maintenant :

```
1  app = HelloWorld()
2  Gtk.main()
```

Et rien ne vous empêche d'en faire plus !

```
1  app = HelloWorld()
2  app_2 = HelloWorld()
3
4  Gtk.main()
```

Maintenant que vous avez votre moule, vous pouvez faire autant de gâteaux que vous voulez.

7.2. Comment l'utiliser ?

Voilà, cette partie est presque terminée. Elle était en effet très courte, la prochaine partie sur Glade viendra la compenser.
Pour terminer, j'aimerais que l'on s'attarde sur comment bien utiliser la POO. Il existe en effet plusieurs façons de résoudre un problème dans une interface graphique, grâce à ce paradigme. Je vais donc vous présenter *ma* vision de la chose. Elle tient en un seul et unique mot, **décomposez**.
Une classe héritant de `Gtk.Window` peut être extrêmement lourde si on ne la décompose pas. Il est donc important d'identifier les différentes *parties* de votre logiciel et donc de réfléchir à l'interface avant, sur papier. Évidement, ces conseils ne s'appliquent que pour des interfaces lourdes, comme celle de la figure suivante par exemple.

FIGURE 7.1. – Un exemple d'interface complexe

Si celle-ci tenait en une seule classe, ce serait imbuvable et infecte d'effectuer la moindre modification. En fait, elle tient en 3 parties – et donc trois classes dans le code – qui interagissent entre elles. Je vous ai représenté les 3 parties sur cette figure :

FIGURE 7.2. – Les 3 parties du programme, de différentes couleurs

Vous voyez donc que j'ai une première partie en vert et une autre en rouge qui sont incluses dans celle en bleu, la fenêtre. Le plus dur étant de réussir à les faire interagir entre elles, ce qui viendra avec de l'expérience.
Au final, n'hésitez pas à faire hériter vos classes d'un layout comme `Gtk.Grid` plutôt que d'une fenêtre. Vous ajoutez ensuite ce layout à un autre layout qui lui contient tout ceux de votre fenêtre. Petit exemple :

```
class GrilleAvecBoutons(Gtk.Grid):
    def __init__(self):
        Gtk.Grid.__init__(self)

        btn = Gtk.Button(label="Mon super bouton")
        self.attach(0, 0, 1, 1)
        # exétéra
```

Listing 2 – Fichier n°1 : une grille pour piloter l'application

Comme on a aussi besoin d'un conteneur principal :

```
class MainWindow(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self)

        box = Gtk.Box()
        sublayout = GrilleAvecBouton()

        box.pack_start(sublayout, True, True, 0)
        self.add(box)
```

Listing 3 – Fichier n°2 : la fenêtre qui les gouverne toutes

Ce code n'est évidement pas fonctionnel, c'est juste une petite démonstration.

Voilà vous savez tout !
J'imagine que vous en avez un peu marre de devoir tout coder. Les développeurs de GTK étant, eux aussi, des flemmards (n'est ce pas ce qui fait l'ADN d'un développeur ?) ont codé un constructeur d'interface, Glade. Je vous le présente au prochain chapitre !

8. Prise en main de Glade

Enfin je vais pouvoir vous introduire Glade ! Depuis le début du tutoriel je n'attends que cette partie. Je commençais à en avoir marre d'être réduis à écrire tout mon code !
Glade est un concepteur d'interface pour GTK+. Il va vous permettre de *dessiner* votre logiciel. C'est encore un peu abstrait, mais je vous promet, c'est magique.

8.1. Installation

Encore un programme à installer ! Allez, on se dépêche, plus vite ce sera finit, plus vite on pourra s'amuser !

8.1.1. Sous Linux

Pas très compliqué pour vous, les Linuxiens, faites simplement un

```
1 sudo apt-get install glade
```

Puis... Voilà.

8.1.2. Sous Windows

Pour les windowsiens, vous pouvez télécharger Glade par ici . C'est un installeur, donc rien de nouveau pour vous.

8.2. Présentation

À l'ouverture de Glade devrait apparaître une fenêtre ressemblant un peu à ça :

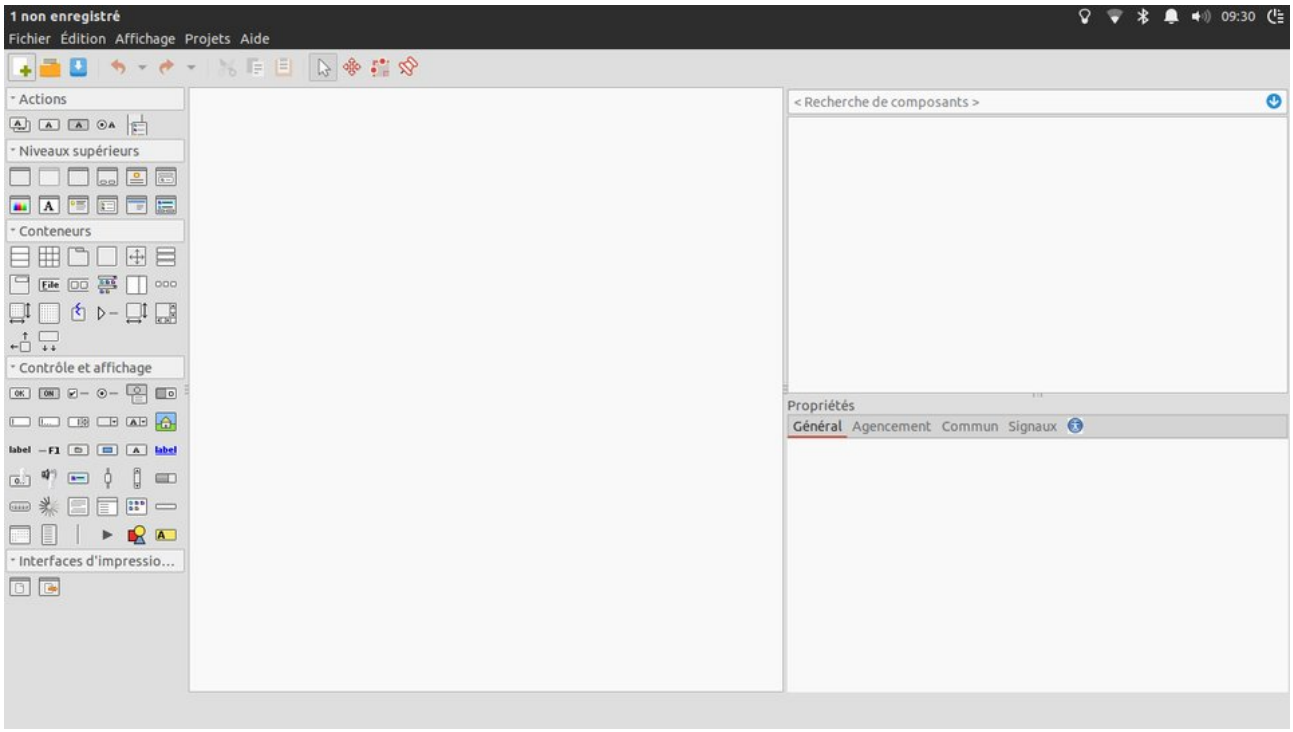

FIGURE 8.1. – Première ouverture de Glade

Comme vous pouvez le voir, il y a pas mal de boutons. Pas de soucis, je vous explique tout. J'ai décomposé la fenêtre en 5 parties. Il est très important de savoir les utiliser. Voyez la figure ci-dessous.

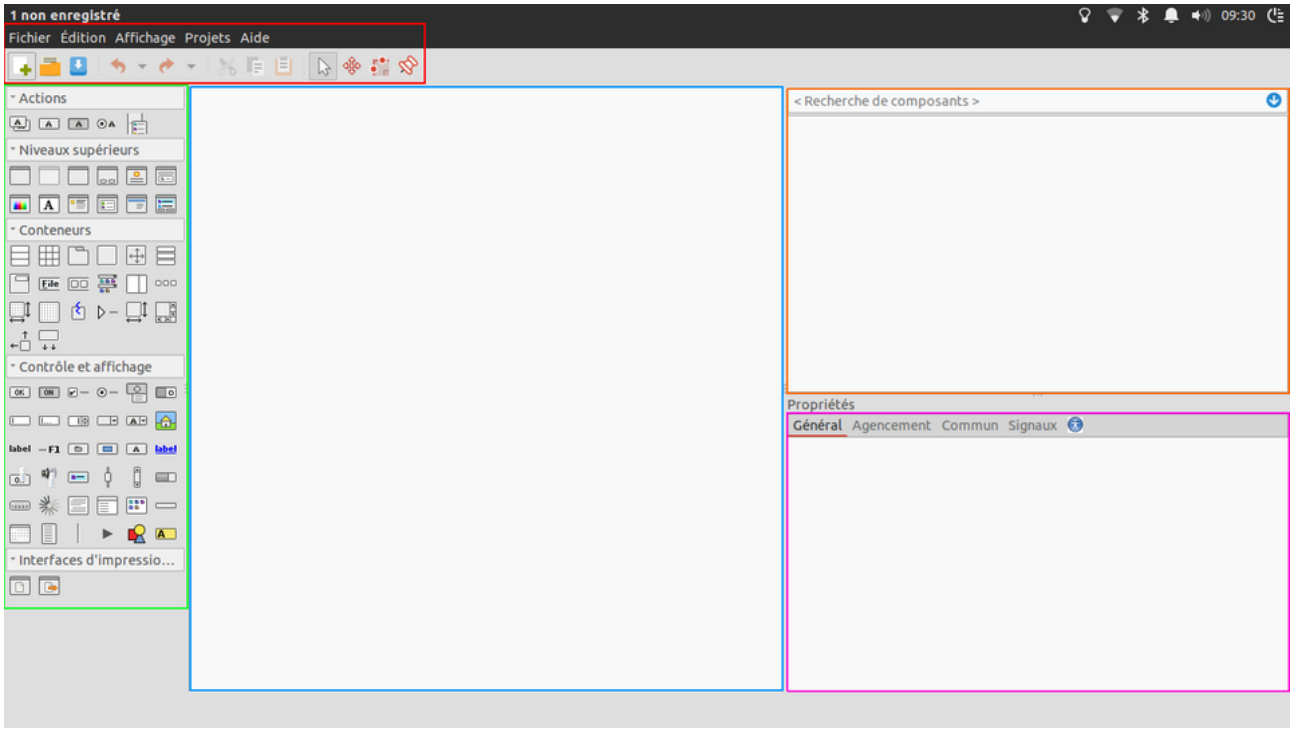

FIGURE 8.2. – Décomposition de Glade

— **En bleu** : C'est la partie centrale, celle que j'appelle *la vue*. C'est ici que vous allez déplacer vos widgets pour créer votre fenêtre.

— **En vert** : Elle ce situe à gauche, elle correspond à celle que j'appelle *l'inventaire*. Elle regroupe les différents widgets de GTK+ implémenté dans Glade. Cette liste s'allonge à chaque mise à jour, il est donc possible que vous ayez plus de widgets que moi !
— **En orange** : Partie que j'appelle *la hiérarchie*, elle représente la hiérarchie de votre fenêtre. Vous y verrez plus clair en l'utilisant.
— **En rose** : C'est ici que vous allez rentrer les propriétés de vos widgets, les fonctions callback à appeler...
— **En rouge** : La barre de menu, pour enregistrer votre projet, lancer un aperçu...

Bien, les présentations étant faites, nous allons commencer par re-faire un *Hello world*, mais avec Glade.
Pour commencer, rendez-vous dans l'inventaire, dans la section *Niveaux supérieurs*. Prenez la première icône et faite la glisser dans la vue ou cliquez dessus. Cette icône représente une `Gtk.Window`. Vous voyez que la hiérarchie se remplie avec votre fenêtre sobrement nommé `window1`. Se trouve à côté en italique le nom de la classe correspondante.

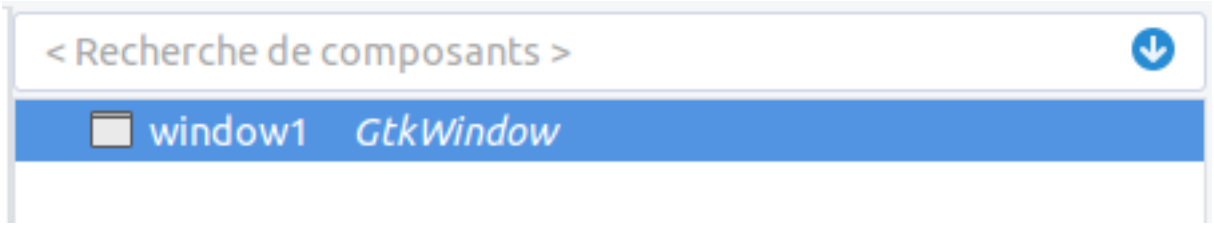

FIGURE 8.3. – Une fenêtre dans la hiérarchie de Glade

Dans la fenêtre des propriétés, juste dessous celle de la hiérarchie, se sont affichées diverses options. Ce sont toutes les propriétés de votre fenêtre. L'onglet *Général* ne concerne que le widget sélectionné. Un widget de type `Gtk.Button` n'aura pas du tout les mêmes options.
Le nom du widget est important. En effet, c'est grâce à celui-ci que nous allons pouvoir les appeler dans notre code. Vous en conviendrez, `window1` n'est pas génial. Pour le changer, Rendez-vous dans l'onglet *Général* et changez sont *Identifiant*. Mettez, par exemple, `main_window`.
Des multitudes d'options sont présentes, différentes pour chaque widget. Je ne vais donc pas pouvoir toutes vous les présenter, autrement ce cours n'en finirait jamais. À vous d'explorer à chaque fois à quoi correspond chaque option.
Vous pouvez ici définir pour notre fenêtre son titre, sa taille minimale, si elle est supprimable (attention, ça ne fermera pas la boucle `Gtk.main()`), si elle est redimensionnable...
Rajoutons maintenant un label. Rendez-vous dans la section *Contrôle et affichage* et chercher une icône avec écrit *label* en noir, ne prenez pas celle en bleu qui correspond à un bouton de lien. Prenez cette icône et glissez-la dans notre `main_window`.
La fenêtre de ma hiérarchie vous indique clairement que ce nouveau label appartient à la fenêtre. Renommez-le.
Pour changer le texte du label, rendez-vous dans la section **Apparence** de l'onglet *Général* et changer son *Étiquette*. Si vous souhaitez utiliser Pango (et donc les balises) cocher la case *Utiliser un balisage*.
Pour voir à quoi ressemble votre fenêtre, allez dans la barre de menu et cherchez une icône ressemblant à 3 engrenages. Cliquez dessus. Voilà qu'un prévisualisateur apparaît. Si elle n'apparaît pas, c'est que quelque chose ne va pas dans votre interface. Vérifiez vos options.
Normalement, il ne devrait pas y avoir de problème et une fenêtre ressemblant à ça apparaît :

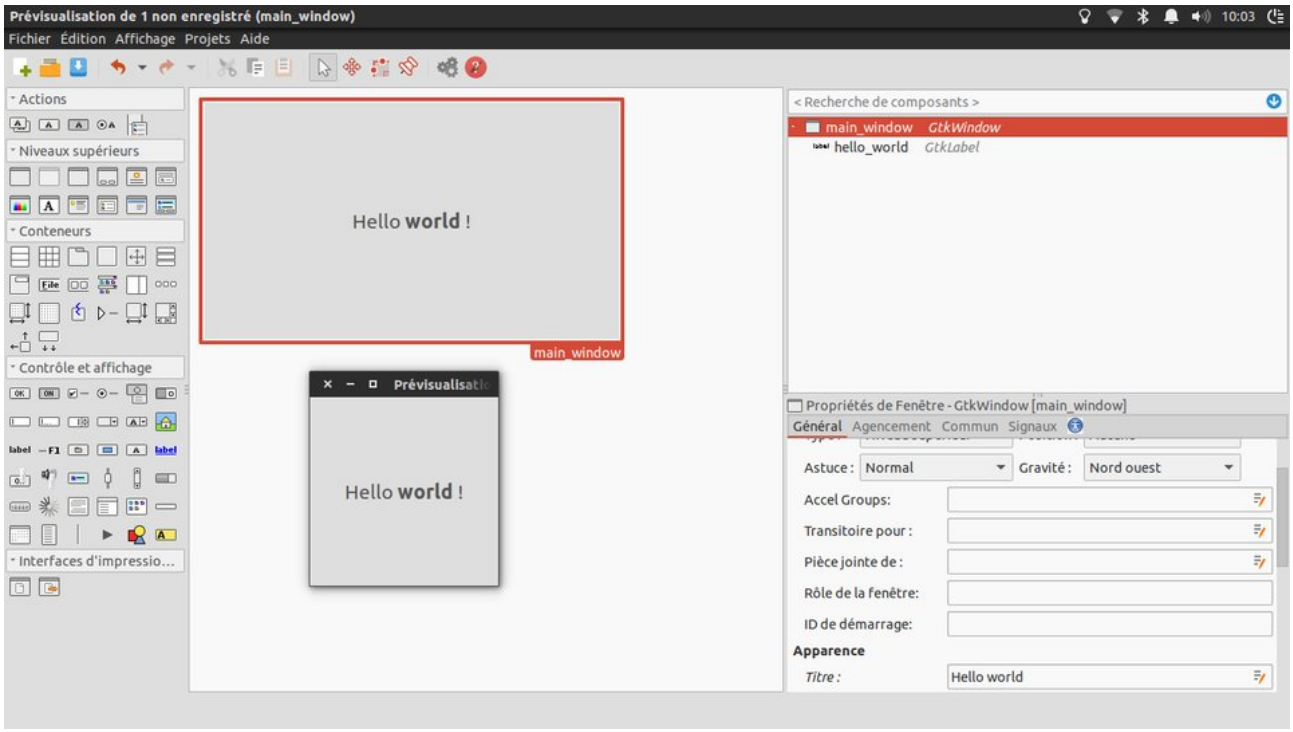

FIGURE 8.4. – Un hello world dans Glade

Je vous laisse jouer un peu avec Glade. Essayer d'ajouter un bouton, explorez les options des widgets, des layouts...
Petite précision quand même, l'onglet *Commun* contient toutes les propriétés qui sont communes à tous les objets. N'hésitez pas à y jeter un coup d'œil.
Bon courage. (Je ne vous abandonne pas, ne vous inquiétiez pas !)

8.3. Utiliser Glade avec Python

C'est bien beau tout ça, faire des fenêtres avec Glade, mais si on ne peut pas l'utiliser avec Python, ça ne nous sers à rien, n'est-ce pas ?

8.3.1. Le design de la fenêtre

Tout d'abord, essayer de faire une fenêtre avec un bouton et un label. Quelque chose comme ça :

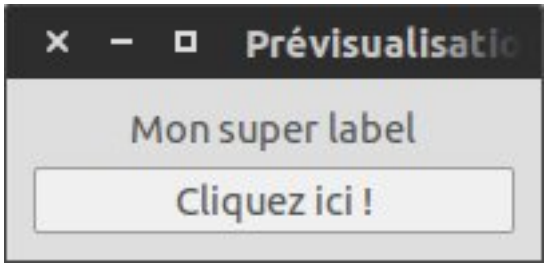

FIGURE 8.5.

Quelques informations :

— Pour ajouter des bordures à la fenêtre, sélectionnez-la et rendez vous dans l'onglet *Commun*, vous y trouverez, tout en bas de cet onglet, une option nommé *Largeur de la bordure*. J'ai rentré 10 comme valeurs.
— Pour tout les layouts, vous trouverez dans l'onglet *Général* une option *Espacement* qui permet de gérer l'espacement entre chaque widgets. Je met toujours 6 pixels.
— Pour changer le label d'un bouton, descendez dans l'onglet *Général* jusqu'à trouver *Étiquette avec image optionnelle* et rentrez le texte dans le cadre qui se trouve en dessous.

Ce que nous voulons faire c'est que quand l'utilisateur clique sur le bouton, le label change de texte. Nous commencerons d'abord sans POO. Puis je vous montrerais avec la POO car il y a deux façon de faire.
Tout d'abord il va falloir définir les signaux. Sélectionnez votre bouton et rendez-vous dans l'onglet *Signaux*. Vous avez ici la liste de tous les signaux qu'un bouton peut émettre. Sélectionnez *clicked* qui est le signal qui nous intéresse. Double cliquez sur *<Type Here>* afin d'entrer le nom du callback (en français, ce serait *gestionnaire*), c'est à dire de la fonction qui va recevoir le signal. Je vais rentrer `on_clicked`.

?

Comment je fais pour passer mon label en argument de cette fonction si je veux pouvoir le changer ?

Justement, cliquez sur le *<Click here>* de la colonne *Données utilisateurs*. Vous allez sélectionner dans la fenêtre qui vient d'apparaître le widget que vous voulez passer en argument à votre gestionnaire. Dans notre cas, le label.
Bien, maintenant enregistrez votre fichier avec *Fichier > Enregistrer*. Je l'enregistre dans mon dossier Python sous le nom `hello.glade`. Voici à quoi ressemble mon fichier :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Generated with glade 3.18.3 -->
3  <interface>
4    <requires lib="gtk+" version="3.12"/>
5    <object class="GtkWindow" id="main_window">
6      <property name="can_focus">False</property>
7      <property name="border_width">10</property>
8      <property name="title" translatable="yes">Hello
           world</property>
9      <child>
10       <object class="GtkBox" id="box_layout">
11         <property name="visible">True</property>
12         <property name="can_focus">False</property>
13         <property name="orientation">vertical</property>
14         <property name="spacing">6</property>
15         <child>
16           <object class="GtkLabel" id="label">
17             <property name="visible">True</property>
18             <property name="can_focus">False</property>
19             <property name="label" translatable="yes">Mon super
                   label</property>
20           </object>
21           <packing>
```

```
22              <property name="expand">False</property>
23              <property name="fill">True</property>
24              <property name="position">0</property>
25            </packing>
26          </child>
27          <child>
28            <object class="GtkButton" id="click_me">
29              <property name="label" translatable="yes">Cliquez ici
                    !</property>
30              <property name="visible">True</property>
31              <property name="can_focus">True</property>
32              <property name="receives_default">True</property>
33              <signal name="clicked" handler="on_clicked"
                    object="label" swapped="no"/>
34            </object>
35            <packing>
36              <property name="expand">False</property>
37              <property name="fill">True</property>
38              <property name="position">1</property>
39            </packing>
40          </child>
41        </object>
42      </child>
43    </object>
44  </interface>
```

Comme vous le voyez, c'est du xml. Vous pouvez l'enregistrer dans un fichier `.glade` et l'ouvrir dans Glade pour avoir exactement la même fenêtre que la mienne ou pour voir les différentes options que j'ai mis dans ma fenêtre.
On peut maintenant passer à la partie code, vous allez voir l'efficacité de Glade !

8.3.2. Le code

Ouvrez un nouveau fichier python et rentrez-y ce qui suit :

```
1  #!/usr/bin/env python3
2  # coding: utf-8
3
4  from gi.repository import Gtk
5
6  def when_button_is_clicked(label):
7      '''
8      Quand le bouton est cliqué
9      '''
10     label.set_text('Hello world!')
11
12
13 builder = Gtk.Builder()
```

```
14 builder.add_from_file('hello.glade')  # Rentrez évidemment votre
       fichier, pas le miens!
15
16 window = builder.get_object('main_window')
17 # Peut se faire dans Glade mais je préfère le faire ici, à vous de
       voir
18 window.connect('delete-event', Gtk.main_quit)
19
20 # Le handler
21 handler = {'on_clicked': when_button_is_clicked}
22 builder.connect_signals(handler)
23
24 window.show_all()
25 Gtk.main()
```

Oui c'est un peu spécial. Décortiquons tout ça.
Il nous faut quelque chose capable de lire notre fichier `.glade`. Le `Gtk.Builder` est là pour ça. Comme son nom l'indique, il va nous aider à construire notre interface. On sélectionne notre fichier avec `Gtk.Builder.add_from_file()`.
Ensuite on récupère notre `Gtk.Window` grâce à son identifiant avec la méthode `Gtk.Builder.get_object()`. En la récupérant, on a aussi toutes les options de notre fenêtre, vous n'avez donc pas à vous occuper de ça ! Ça correspond à faire ça :

```
1 window = Gtk.Window(title='Hello world')
2 window.set_...
```

Vous en conviendrez, c'est plus simple avec Glade !
On passe ensuite aux handlers. Ici j'ai créé un dictionnaire qui contient en indice le nom du signal et en valeur le callback à appeler quand ce signal est émit. Ici, notre dictionnaire ne contient qu'un seul handler, mais dans une fenêtre plus complexe, on peut facilement monter à une dizaine de signaux. Il existe un autre moyen de définir ses handlers en passant par une classe. Mais n'utilisant jamais ce système, je vous laisse regarder ici pour plus de détail .
Puis on indique au `Gtk.Builder` d'utiliser ce dictionnaire avec `Gtk.Builder.connect_signals()`. Petite précision, certains d'entre vous auront peut être remarqué que je ne prend pas de bouton en argument de mon callback. Et c'est bien vu. En fait, quand vous passez un argument avec Glade, le bouton n'est plus passé en argument automatiquement. Si vous souhaitez passer plus d'un seul argument à un callback, il va falloir passer par Python et oublier Glade pour connecter votre widget.
Je n'ai pas à vous expliquer le reste, rien ne devant vous poser problème ici !

Je vous l'avais bien dit, Glade est magique ! Vous pouvez désormais vous lancer dans la création de n'importe quel programme, rien ne devrais vous bloquer. Il nous reste deux trois notions à aborder qui sont plus complexes. Courage !

Bon bah voilà... Je n'ai plus grand chose à vous apprendre maintenant. Ne partez pas si vite voyons ! Il nous reste encore un gros TP à faire, ce ne serais pas drôle sinon !

Waouh, on en aura parcouru du chemin ensemble !

Vous voilà capable de créer des interfaces graphiques, de la plus simple à la plus compliquée. Vous en voulez-encore ? Explorez les différents widgets que vous propose PyGObject, ou créez carrément le vôtre ↗ !