CM6: interfaces graphiques (suites)

Programmation 3: L3 mathématique

Florian Bridoux

November 21, 2019

Événements souris dans darea

On importe une autre librairie, Gdk :

```
from gi.repository import Gtk, Gdk
```

On autorise darea à recevoir les événements souris :

```
darea.set_events(Gdk.EventMask.BUTTON_PRESS_MASK |
                 Gdk.EventMask.BUTTON_RELEASE_MASK |
                 Gdk.EventMask.BUTTON_MOTION_MASK )
```

Événements souris dans darea

On définit ensuite 3 callbacks :

```
def darea_on_button_press (darea, event) :
    print("Bouton enfoncé en", event.x, event.y)

def darea_on_button_release (darea, event) :
    print("Bouton relaché en", event.x, event.y)

def darea_on_motion_notify (darea, event) :
    print("Souris déplacée en", event.x, event.y)

darea.connect("button-press-event", darea_on_button_press)
darea.connect("button-release-event", darea_on_button_release)
darea.connect("motion-notify-event", darea_on_motion_notify)
```

Événements souris dans darea

On peut leur rajouter un 3e paramètre data, ou rajouter des attributs à darea. Exemple :

```
def darea_on_button_press (darea, event) :
    print("Bouton enfoncé en", event.x, event.y)
    darea.coord_x = event.x
    darea.coord_y = event.y
    darea.queue_draw()

def darea_on_draw (darea, cr) :
    print("Événement draw reçu")
    x = darea.coord_x
    y = darea.coord_y
    cr.arc (x, y, 40, 0, 2*math.pi) # centre, rayon, angles
    cr.set_source_rgb(0.8, 0.8, 0.8)
    cr.fill_preserve()
    cr.set_source_rgb(0, 0.8, 0)
    cr.stroke()

darea.coord_x = 200
darea.coord_y = 200
```

Événements clavier dans darea

On a besoin de la librairie Gdk :

```
from gi.repository import Gtk, Gdk
```

On autorise darea a "prendre le focus clavier", c'est-à-dire à recevoir les événements clavier :

```
darea.set_can_focus(True)
```

On donne le focus clavier à darea :

```
darea.grab_focus()
```

Enfin on relie une fonction darea_on_key_press à l'événement "key-press-event" :

```
darea.connect("key-press-event", darea_on_key_press)
```

La fonction darea_on_key_press sera appelée pour chaque touche enfoncée.

Événements clavier dans darea

On a besoin de la librairie Gdk :

```
from gi.repository import Gtk, Gdk
```

On autorise darea a "prendre le focus clavier", c'est-à-dire à recevoir les événements clavier :

```
darea.set_can_focus(True)
```

On donne le focus clavier à darea :

```
darea.grab_focus()
```

Enfin on relie une fonction darea_on_key_press à l'événement "key-press-event" :

```
darea.connect("key-press-event", darea_on_key_press)
```

Événements clavier dans darea

La fonction darea_on_key_press sera appelée pour chaque touche enfoncée :

```
def darea_on_key_press (darea, ev) :
    print ("Gdk.KEY_", Gdk.keyval_name(ev.keyval), sep='')
    if ev.keyval == Gdk.KEY_a :
        print("c'est la touche 'a'")
    elif ev.keyval == Gdk.KEY_space :
        print("c'est la touche espace")
```

De la même manière on peut relier une fonction darea_on_key_release à l'événement "key-release-event".

```
darea.connect("key-release-event", darea_on_key_press)
```

Remarque: chaque fois que l'on clique sur un bouton, il prend le focus clavier, et donc darea le perd. On peut redonner le focus à darea par exemple dans darea_on_button_press.

Titre: Gestion des données

Nombreux widgets $\rightarrow$ nombreuses callback connectées à des signaux
Ces callback ont besoin d'accéder aux données du programme, les modifier ou d'agir sur d'autres widgets (par exemple un bouton sur darea).

Titre: Gestion des données

Solution 1: mémoriser les données intéressantes en attributs de certains widgets.

Exemple:

```
bouton1 = Gtk.Button(label="Vider")
bouton1.connect ("clicked", bouton1_on_clicked)
darea = Gtk.DrawingArea()
darea.connect("draw", darea_on_draw)
darea.coord_x = 10
darea.coord_y = 20
bouton1.darea = darea

def bouton1_on_clicked (widget) :
    darea = widget.darea
    darea.queue_draw()    # pour forcer un réaffichage

def darea_on_draw (darea, cr) :
    cr.arc (darea.coord_x, darea.coord_y, rayon, 0, 2*math.pi)
    cr.stroke()
```

Titre: Gestion des données

- gros désordre dans le programme (savoir dans quel widget est stocké quelle information ; problèmes de redondance possible)
- problème de couplage des widgets (des widgets devront connaître d'autres widgets ; problème d'ordre de déclaration)
- rend difficile l'évolution du code.

Titre: Gestion des données

Solution 2 plus propre :

- regrouper les données du programme dans une classe MyData
- mémoriser en attribut dans my les widgets intéressants (darea, etc)
- choix:
  - mémoriser my dans chaque widget $\rightarrow$ rapidement lourd
  - passer my en 3e paramètre de chaque connect $\rightarrow$ il sera reçu en paramètre supplémentaire dans chaque callback.

Titre: Gestion des données

Exemple:

```
class MyData:
    def __init__(self) :
        self.coord_x = 10
        self.coord_y = 20

my = MyData()
bouton1 = Gtk.Button(label="Vider")
bouton1.connect ("clicked", bouton1_on_clicked, my)
my.darea = Gtk.DrawingArea()
my.darea.connect("draw", darea_on_draw, my)

def bouton1_on_clicked (widget, my) :
    my.darea.queue_draw()    # pour forcer un réaffichage

def darea_on_draw (darea, cr, my) :
    cr.arc (my.coord_x, my.coord_y, rayon, 0, 2*math.pi)
    cr.stroke()
```

Widget Label

Le Widget Label permet d'afficher du texte. Usage:

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

win = Gtk.Window()
win.set_size_request(400, 300) #taille: largeur,hauteur
win.set_title("Exemple de label")
win.connect("destroy", Gtk.main_quit)

label1 = Gtk.Label(label="texte label 1")
label2 = Gtk.Label(label="texte label 2")
label3 = Gtk.Label(label="texte label 3")

vbox = Gtk.VBox()
vbox.pack_start(label1, expand=False, fill=False, padding=0)
vbox.pack_start(label2, expand=False, fill=False, padding=0)
vbox.pack_start(label3, expand=False, fill=False, padding=0)

win.add(vbox)
win.show_all()
Gtk.main()
```

Exercice d'application

Écrire un programme qui crée une fenêtre avec un bouton et un label. Le label contiendra la texte "J'ai cliqué *n* fois sur le bouton" en remplaçant *n* par ce nombre de fois.
Vous pouvez utiliser la méthode .set_text() de la class Label pour modifier le text du label.

Exercice d'application

Écrire un programme qui crée une fenêtre avec un bouton et un label. Le label contiendra la texte "J'ai cliqué *n* fois sur le bouton" en remplaçant *n* par ce nombre de fois.
Vous pouvez utiliser la méthode .set_text() de la class Label pour modifier le text du label.

voir correction

Champ de saisie Entry

Ce widget permet de saisir une ligne de texte.

Usage :

```
entry = Gtk.Entry()
entry.connect("activate", entry_on_activate)

def entry_on_activate(widget) :
    print("Entry :", widget.get_text())
```

Le signal "activate" est envoyé chaque fois que l'utilisateur appuie sur [Entrée].

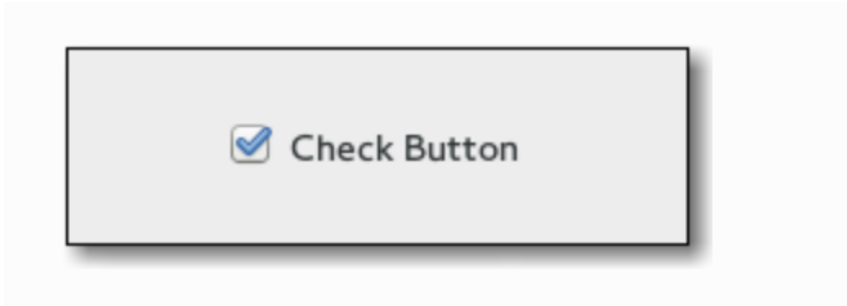
Champ de saisie Entry

On précède souvent ce widget d'un Label, avec un peu d'espace autour. Exemple:

```
label2 = Gtk.Label("Panier :")
hbox.pack_start(label2, expand=False, fill=False, padding=5)

entry1 = Gtk.Entry()
hbox.pack_start(entry1, expand=False, fill=False, padding=0)
entry1.connect("activate", entry1_on_activate, my)

def entry1_on_activate(widget, my) :
    my.panier = widget.get_text()
    my.darea.queue_draw()
```

CheckButton

Le widget CheckButton permet de dessiner une case à cocher avec un nom.



Usage :

```
check = Gtk.CheckButton(label="nom du bouton")
check.set_active(etat_booleen)
check.connect("toggled", check_on_toggled)

def check_on_toggled(widget) :
    print("État :", widget.get_active())
```

Chaque fois que son état change, la callback est appelée.

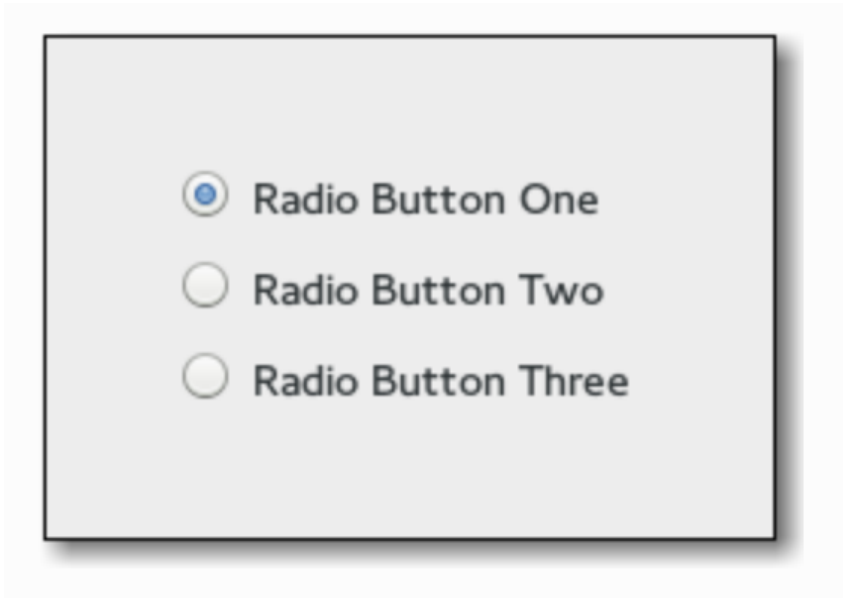
CheckButton

Exemple:

```
class MyData:
    def __init__(self) :
        ...
        self.rempli = True

check1 = Gtk.CheckButton(label="Rempli")
check1.set_active(my.rempli)
hbox.pack_start(check1, expand=False, fill=False, padding=10)
check1.connect("toggled", check1_on_toggled, my)

def check1_on_toggled(widget, my) :
    my.rempli = widget.get_active()
    my.darea.queue_draw()

def darea_on_draw (darea, cr, my) :
    ...
    cr.arc (my.coord_x, my.coord_y, my.rayon, 0, 2*math.pi)
    if my.rempli :
        cr.set_source_rgb(0.8, 0.8, 0.8)
        cr.fill_preserve()
    cr.set_source_rgb(0, 0.8, 0)
    cr.stroke()
```

Bouton radio: RadioButton

Les boutons radio fonctionnent en groupe : un seul bouton est enfoncé à la fois.



Bouton radio: RadioButton

Exemple:

```
rad1 = Gtk.RadioButton.new_with_label_from_widget(None, "Choix 1")
rad1.connect("toggled", radio_on_toggled)
rad2 = Gtk.RadioButton.new_with_label_from_widget(rad1, "Choix 2")
rad2.connect("toggled", radio_on_toggled)
...

def radio_on_toggled (widget) :
    print ("Radio", widget.get_label(), widget.get_active())
```

Pour créer un groupe de boutons radios, on passe None au premier, puis l'un des widgets déjà créés aux autres.
Lorsqu'on enfonce un bouton radio cela provoque deux signaux "toggled":

- 1 pour le bouton enfoncé : widget.get_active() est True
- 1 pour le bouton relaché : widget.get_active() est False

Bouton radio: RadioButton

Exemple de création avec une boucle :

```
class MyData:
    def __init__(self) :
        ...
        self.panier = "Rien"

prec = None
for nom in ["Bijou", "Caillou", "Chou"] :
    tmp = Gtk.RadioButton.new_with_label_from_widget(prec, nom)
    hbox.pack_start(tmp, expand=False, fill=False, padding=0)
    tmp.connect("toggled", radio_on_toggled, my)
    prec = tmp

def radio_on_toggled (widget, my) :
    print ("Radio", widget.get_label(), widget.get_active())
    if widget.get_active() :
        my.panier = widget.get_label()
        my.darea.queue_draw()

def darea_on_draw (darea, cr, my) :
    ...
    cr.show_text("Panier : "+my.panier)
    ...
```

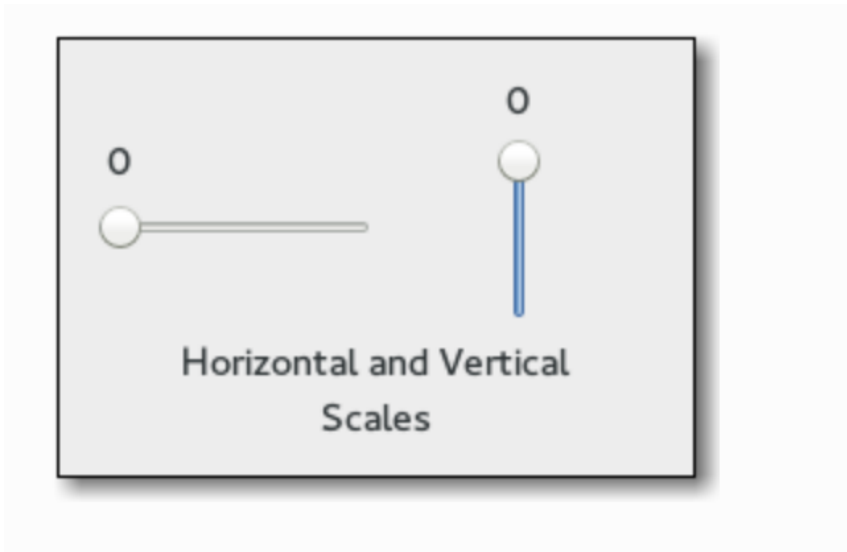
Bouton radio: RadioButton

Il peut être utile de stocker des données en attribut dans les boutons radio si on veut les utiliser par la suite :

```
for nom in [("Bijou", "une bague en or"),
            ("Caillou", "un vulgaire diamant"),
            ("Chou", "un choux rouge")] :
    tmp = Gtk.RadioButton.new_with_label_from_widget(prec, nom[0])
    tmp.infos = nom[1]
    ...

def radio_on_toggled (widget, my) :
    if widget.get_active() :
        my.panier = widget.infos
        ...
```

Slider Scale

Le widget Scale permet de dessiner un slider (ou potentiomètre) horizontal ou vertical.



Slider Scale

Usage :

```
scale = Gtk.Scale.new_with_range(orientation, min, max, step)
scale.set_value(value)
scale.connect("value-changed", scale_on_value_changed)

def scale_on_value_changed(widget) :
    print("Scale value :", widget.get_value())
```

L'orientation est Gtk.Orientation.HORIZONTAL ou Gtk.Orientation.VERTICAL.
Ce widget a tendance à prendre le moins de place possible -> on lui donne souvent une taille minimale :

```
scale.set_size_request(largeur_min,-1)  # largeur, hauteur (-1 pour unset)
```

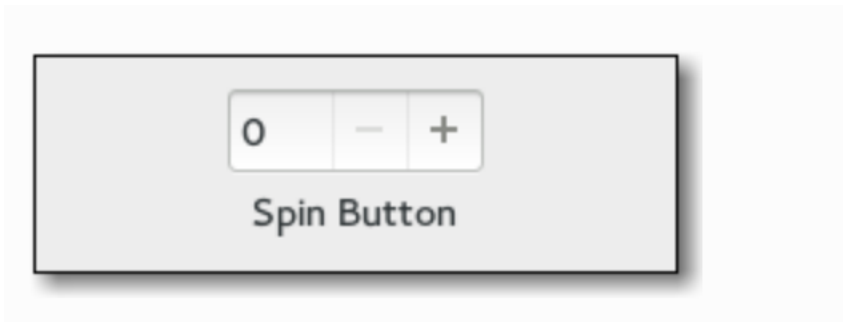
Slider Scale

On précède souvent ce widget d'un Label, avec un peu d'espace autour. Exemple:

```
class MyData:
    def __init__(self) :
        ...
        self.rayon = 20

label1 = Gtk.Label("Rayon :")
hbox.pack_start(label1, expand=False, fill=False, padding=10)

scale1 = Gtk.Scale.new_with_range(Gtk.Orientation.HORIZONTAL, 10, 100, 1)
scale1.set_value(my.rayon)
scale1.set_size_request(100,-1)
hbox.pack_start(scale1, expand=False, fill=False, padding=0)

scale1.connect("value-changed", scale1_on_value_changed, my)

def scale1_on_value_changed(widget, my) :
    print("Scale1 :", widget.get_value())
    my.rayon = widget.get_value()
    my.darea.queue_draw()
```

Slider Scale

On peut aussi utiliser à la place un SpinButton.



Voir documentation.

Exercice

Réalisez un programme python qui affiche un triangle et permet de le déformer en bougeant ses sommets.