Paradigmas de Programación

Práctica II - Curso 2016/17

Guía y Consejos útiles (I)

Consejos generales

- No es necesario que todos los controles de la aplicación estén definidos o creados en el fichero generado por Glade. Es perfectamente posible crear nuevos controles en tiempo de ejecución, y de hecho ese es el enfoque más conveniente para esta práctica, ya que lo lógico es que los controles que representan o forman parte de los coches se creen cuando pasemos a un nuevo nivel. También es perfectamente posible (aunque no recomendable) crear la aplicación sin usar Glade.
- Para tener un referencia de un control definido en Glade y poder cambiar sus propiedades, se debe usar el método *get_object* del objeto de clase *Builder* que usamos para leer el fichero:

```
class Aplicacion:
    def __init__(self):
        self.builder = gtk.Builder()
        self.builder.add_from_file("fichero.glade")
        ...
        self.control = self.builder.get_object("nombre_control_glade")
        ...
```

  El nombre del control en Glade se establece en la pestaña "General", campo "Name".
- **Atención**: Todos los controles que se crean por programa son, por defecto, **invisibles**. Para hacer que sean visibles se debe llamar al método *show()*. En Glade todos los controles se crean como visibles **salvo la ventana principal**. Se debe seleccionar la ventana principal e ir a la pestaña "Common", propiedad "Visible" y hacer que muestre "Yes".
- **Se puede asignar un mismo código para que trate un mismo tipo evento de varios controles**. El método *control.connect(nombre_evento, código_evento, dato)* permite indicar un valor (el parámetro *dato*) que se incluirá como último parámetro en la llamada a *código_evento* y que puede usarse como identificador del control sobre el que se ha producido el evento (también podría usarse el parámetro *widget* pero con este método te llevas menos sorpresas). De esta forma una única función puede tratar eventos de muchos controles distintos.

Información específica

- **Creación de la interfaz de usuario**: El método más rápido de crear una interfaz de usuario aceptable es anidar contenedores **VPaned** y **HPaned**, cada uno de ellos permite añadir 2 controles o contenedores y la situación normal es que uno de ellos tenga tamaño fijo (la etiqueta de título, por ejemplo) y el otro cambie de tamaño si varían las dimensiones de la aplicación (el área de dibujo, por ejemplo). En Glade esto se consigue haciendo que la parte fija tenga el parámetro "Redimensionar" (pestaña Packing) a No y la parte variable lo tenga como Yes. (el parámetro "Encoger" siempre debería estar como Yes).

Para el área principal de juego se recomienda usar un contenedor **gtk.Fixed**.

- **Uso de un contenedor de posición fija**: El contenedor **gtk.Fixed** permite disponer controles cuya posición (en pixels) podemos fijar o cambiar en tiempo de ejecución.

```
cont_fixed = GtkFixed() # Crear un contenedor de tipo "fixed"
cont_fixed.put(control, x, y) # Añadir control en posición (x,y)
cont_fixed.move(control, x, y) # Mover control existente a (x,y)
cont_fixed.remove(control) # Elimina el control del contenedor
cont_fixed.get_children() # Devuelve una lista de todos los
                          # controles que están en el contenedor
# Establecer las dimensiones del control (pixels)
control.set_size_request(lx, ly)
```

  Las coordenadas se expresan en pixels y son relativas al contenedor, no a la pantalla. Cuidado porque este contenedor, a diferencia de los otros, no ajusta automáticamente el tamaño de los controles que se le añaden, es necesario que lo hagamos nosotros mediante el método *set_size_request*

- **Activar/desactivar un control**: Se llama al método *set_sensitive(bool)* del control.
- **Cambiar el tipo de letra y tamaño de fuente de una etiqueta (gtk.Label):** La forma más sencilla es usar el lenguaje de marcas al establecer el texto. Ejemplo:

```
etiqueta.set_markup("<span font_dec='Tahoma 25'><b>"+texto+"</b></span>"
```

- **Cambiar el color de fondo de un contenedor (gtk.Frame, gtk.EventBox, etc)**

```
caja.modify_bg(gtk.STATE_NORMAL, caja.get_colormap().alloc_color(color))
```

  Donde *caja* representa el control y *color* es una cadena de texto con el nombre de uno de los colores estándar de X11, acceder al siguiente enlace para consultarlos:

  https://en.wikipedia.org/wiki/X11_color_names

- **Cambiar el color de fondo de un botón (gtk.Button):** El proceso es un poco más complicado que el explicado anteriormente, ya que hay que cambiar el estilo:

```
col = boton.get_colormap().alloc_color(color)
estilo = boton.get_style().copy()
estilo.bg[gtk.STATE_NORMAL] = col
estilo.bg[gtk.STATE_SELECTED] = col
boton.set_style(estilo)
```

- **Uso del contenedor gtk.Notebook para representar múltiples "pantallas"**: Es posible diseñar la aplicación usando una única ventana, sin mostrar cuadros de diálogo, utilizando un contenedor de tipo gtk.Notebook (que permite diseñar distintos paneles seleccionables mediante pestañas), ocultando la pestaña (Glade: pestaña "General", campo "Mostrar solapas:" con valor No) y en el programa seleccionando el panel que se quiere mostrar con la llamada al método *set_current_page(n)* del control Notebook (*n* es un entero con valor entre 0 y el máximo de paneles menos uno).
- **Uso de un control imagen + capturar eventos de pulsación y arrastre de ratón**: El control g**tk.Image** permite mostrar imágenes (almacenadas en ficheros de tipo GIF, JPG o PNG), pero como este control no puede capturar eventos se debe incluir en un contenedor g**tk.EventBox**, que si puede hacerlo. Para capturar eventos de "arrastre" de ratón es necesario

habilitar al contenedor. En el código siguiente se muestra cómo hacerlo creando los controles en el programa (sin usar Glade):

```
class Aplicacion:
  def __init__(self):
    ...
    self.caja = Gtk.EventBox()
    self.caja.show()
    self.img = Gtk.Image()
    self.img.show()
    self.img.set_from_file(fichero_gif_jpg_png)
    self.caja.add(self.img)
    self.caja.set_events(gtk.gdk.BUTTON_PRESS_MASK |
                         gtk.gdk.BUTTON_RELEASE_MASK |
                         gtk.gdk.POINTER_MOTION_MASK)
    # dato es un valor de tipo cualquiera definido por ti que se puede
    # usar para identificar este control (caja) de cualquier otro al
    # que conectes con las mismas funciones on_pulsar, on_soltar, ..
    self.caja.connect("button_press_event", self.on_pulsar, dato)
    self.caja.connect("button_release_event", self.on_soltar, dato)
    self.caja.connect("motion_notify_event", self.on_arrastrar, dato)

  def evento_pulsar(self, widget, event, data):
    # Para distinguir arrastrar de mover el ratón:
    self.arrastrando = True
    # En data tienes el dato que pasaste en la llamada
    # a connect y que te puede servir para identificar
    # el control sobre el que se ha pulsado.

    # event.x_root y event.y_root proporcionan las
    # coordenadas (en pixels) respecto a la pantalla
    # principal donde se ha pulsado el ratón.

    # Debes almacenar estas coordenadas ya que vas a trabajar
    # con coordenadas relativas:
    self.x0 = int(event.x_root)
    self.y0 = int(event.y_root)
    ...
    return gtk.TRUE

  def evento_arrastrar(self, widget, event, data):
    # Para no confundir arrastrar con sólo mover el ratón:
    if not self.arrastrando:
      return gtk.FALSE
    # Calculas lo que se ha movido el ratón desde el punto
    # en que se produjo la pulsación (en pixels)
    dx = int(event.x_root) - self.x0
    dy = int(event.y_root) - self.y0
    ...
    return gtk.TRUE

  def evento_soltar(self, widget, event, data):
    self.arrastrando = False
    # Desplazamiento final:
    dx = int(event.x_root) - self.x0
    dy = int(event.y_root) - self.y0
    ...
    return gtk.TRUE
```