Architecture d'un programme interactif graphique

clavier & souris

Objectifs:

- Usage des événements du clavier
- Usage des boutons et du mouvement de la souris

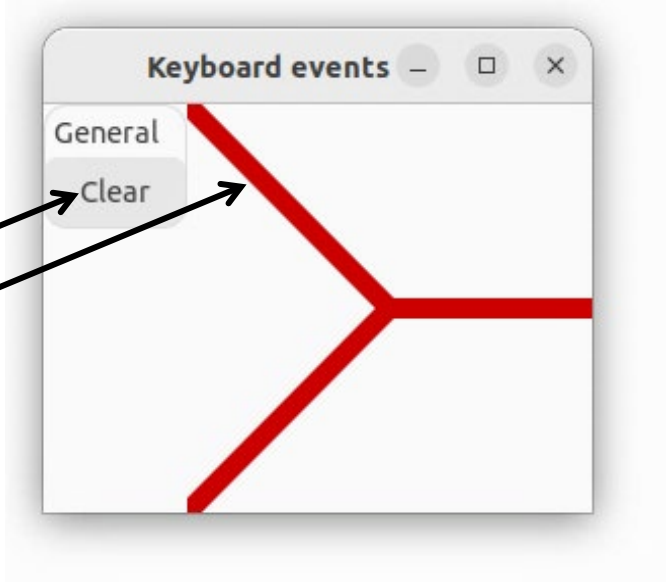
Remarques:

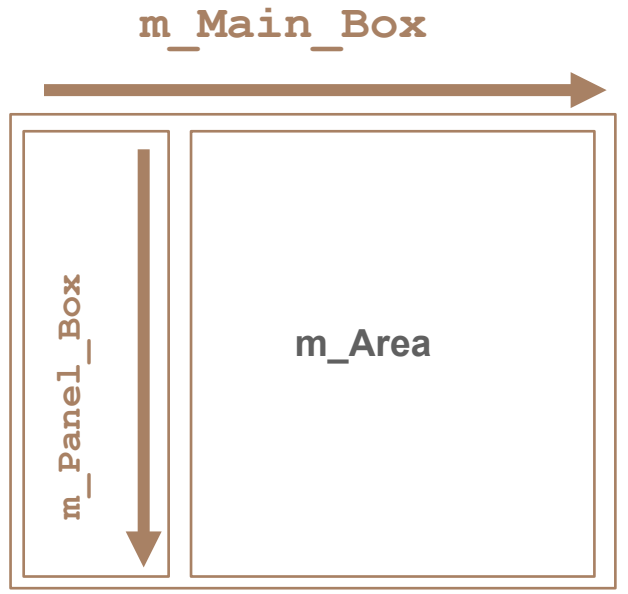
Revoir le cours de la semaine précédente sur la gestion d'événements liés à des boutons
Le code présenté dans ce cours est fourni et détaillé dans la série6 niveau 0.

myevent.h

Exemple1 Clavier (1)

```
6   class MyEvent : public Gtk::Window
7   {
8   public:
9       MyEvent();
10
11  private:
12      // GUI layout
13      Gtk::Box m_Main_Box;
14      Gtk::Box m_Panel_Box;
15      Gtk::Box m_Buttons_Box;
16      Gtk::Frame m_Panel_Frame;
17      Gtk::Button m_Button_Draw_Clear;
18      Gtk::DrawingArea m_Area;
19
20      bool draw ;             // current drawing state
21      bool thick_linewidth ;  // another state variable
22
23      //Button Signal handlers:
24      void on_button_clicked_draw_clear();
25
26      // DrawingArea signal handler:
27      void on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
28                   int width, int height);
29
30      // keyboard event signal handler:
31      bool on_window_key_pressed(guint keyval, guint keycode,
32                                 Gdk::ModifierType state);
33
34  };
```

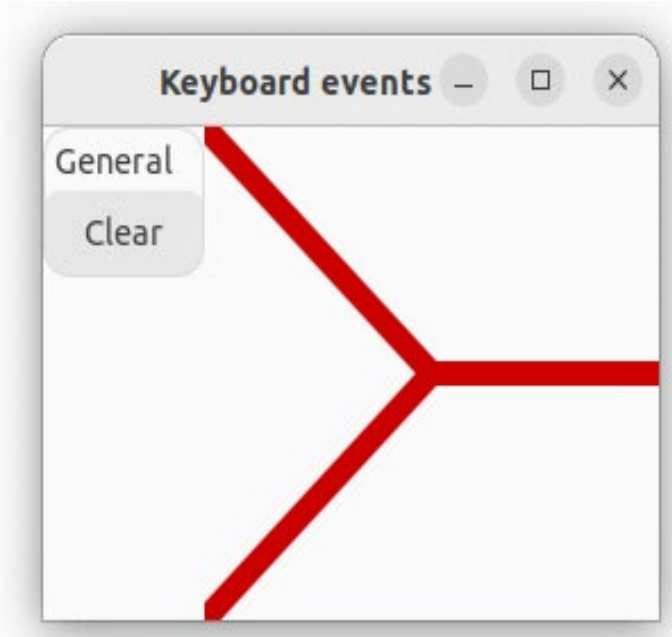




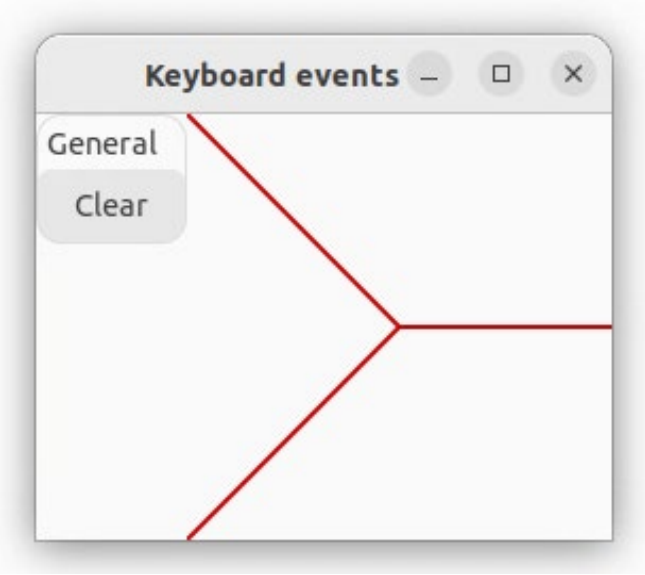
myevent.cc

Clavier (2)

```
37      // Handling Keyboard Events.
38      auto controller = Gtk::EventControllerKey::create();
39      controller->signal_key_pressed().connect(
40                  sigc::mem_fun(*this, &MyEvent::on_window_key_pressed), false);
41      add_controller(controller);
42  }
```




```
44  void MyEvent::on_button_clicked_draw_clear()
45  {
46      std::cout << "changing drawing state through the button" << std::endl;
47      draw = !draw;
48      if(draw)    m_Button_Draw_Clear.set_label("CLEAR!");
49      else        m_Button_Draw_Clear.set_label("DRAW !");
50      m_Area.queue_draw();
51  }
```




```
62          if(thick_linewidth)         cr->set_line_width(10.0);
63          else                        cr->set_line_width(2.0);
```

usage du clavier avec son signal handler :

```
80  bool MyEvent::on_window_key_pressed(guint keyval, guint, Gdk::ModifierType state)
81  {
82      switch(gdk_keyval_to_unicode(keyval))
83      {
84        case 'c':
85        case 'd':
86          std::cout << "changing drawing state through the keyboard" << std::endl;
87          draw = !draw;
88          if(draw)    m_Button_Draw_Clear.set_label("CLEAR!");
89          else        m_Button_Draw_Clear.set_label("DRAW !");
90          m_Area.queue_draw();
91          return true;
92        case 'w':
93          std::cout << " changing state variable thick_linewidth" << std::endl;
94          thick_linewidth= !thick_linewidth;
95          m_Area.queue_draw();
96          return true;
97        case 'q':
98          std::cout << "Quit" << std::endl;
99          hide(); //ou  exit(0);
100         return true;
101     }
102     //the event has not been handled
103     return false;
104 }
```

même action que le bouton m_Button_Draw_Clear

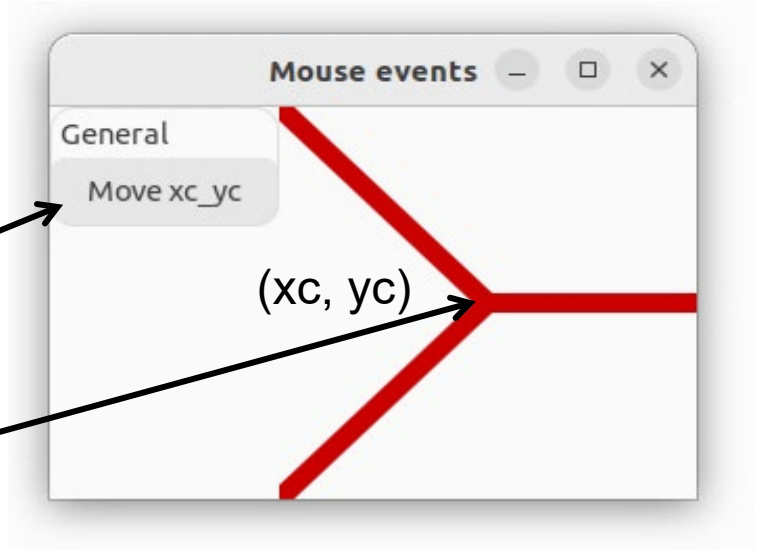
inverse l'état de la variable thick_linewidth

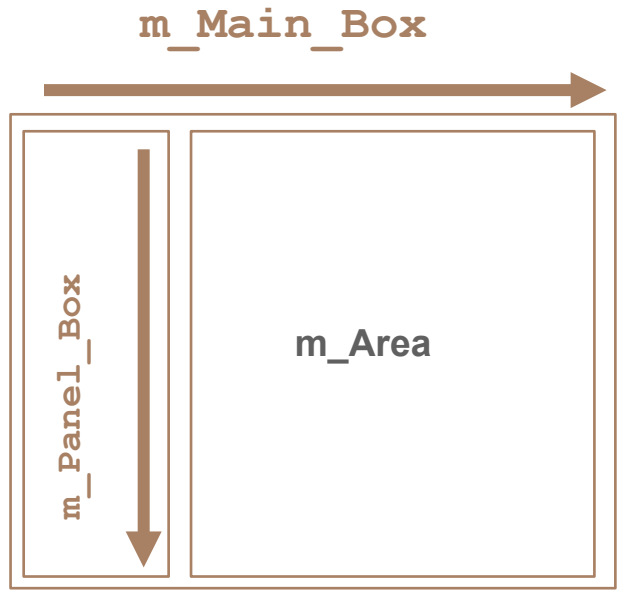
quitte le programme

myevent.h

Exemple2 Souris (1)

```
6   class MyEvent : public Gtk::Window
7   {
8   public:
9       MyEvent();
10
11  private:
12      // GUI layout
13      Gtk::Box m_Main_Box;
14      Gtk::Box m_Panel_Box;
15      Gtk::Box m_Buttons_Box;
16      Gtk::Frame m_Panel_Frame;
17      Gtk::Button m_Button_Move_xc_yc;
18      Gtk::DrawingArea m_Area;
19
20      // used to enable using mouse move data
21      bool move_xc_yc ;
22      int  xc,yc;          // drawing parameter
23
24      //Button Signal handlers:
25      void on_button_clicked_move_xc_yc();
26
27      // DrawingArea signal handler:
28      void on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
29                   int width, int height);
30
31      // mouse event signal handler:
32      void set_mouse_controller();
33      void on_drawing_left_click(int n_press, double x, double y);
34      void on_drawing_right_click(int n_press, double x, double y);
35      void on_drawing_move(double x, double y);
36  };
```





myevent.cc

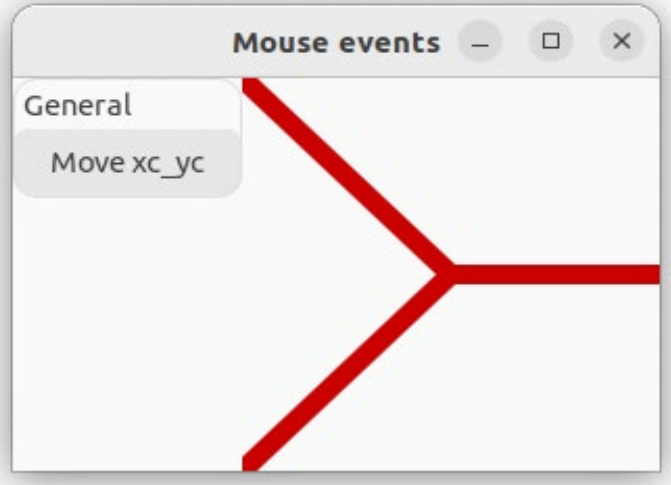
Souris (2)

```
66  void MyEvent::set_mouse_controller()
67  {
68      auto left_click = Gtk::GestureClick::create();
69      auto right_click = Gtk::GestureClick::create();
70      auto move = Gtk::EventControllerMotion::create();
71
72      left_click->set_button(GDK_BUTTON_PRIMARY);
73      right_click->set_button(GDK_BUTTON_SECONDARY);
74
75      left_click->signal_pressed().connect(
76          sigc::mem_fun(*this, &MyEvent::on_drawing_left_click));
77      right_click->signal_pressed().connect(
78          sigc::mem_fun(*this, &MyEvent::on_drawing_right_click));
79      move->signal_motion().connect(
80          sigc::mem_fun(*this, &MyEvent::on_drawing_move));
81
82      m_Area.add_controller(left_click);
83      m_Area.add_controller(right_click);
84      m_Area.add_controller(move);
85  }
```

```
87  void MyEvent::on_drawing_left_click(int n_press, double x, double y)
88  {
89      cout << "mouse left clic detected and processed" << endl;
90      cout << "x = " << x << "    y = " << y << endl;
91      xc = x;
92      yc = y;
93      m_Area.queue_draw();
94  }
```

inverse l'état de la variable move_xc_yc

```
96  void MyEvent::on_drawing_right_click(int n_press, double x, double y)
97  {
98      cout << "mouse right clic detected and processed" << endl;
99      move_xc_yc = !move_xc_yc;
100     if(move_xc_yc ) m_Button_Move_xc_yc.set_label("Do Not Move!");
101     else            m_Button_Move_xc_yc.set_label("Move (xc,yc)");
102     m_Area.queue_draw();
103 }
```




```
mouse left clic detected and processed
x = 73    y = 96
```

signal handler du mouvement de la souris :

```
105  void MyEvent::on_drawing_move(double x, double y)
106  {
107      static char c('A');
108      cout << c << " => mouse move detected and processed" << endl;
109      if(++c >'Z') c= 'A'; // reset
110
111      if(move_xc_yc)
112      {
113          if(x > 0 and x < m_Area.get_width())
114              xc = x;
115
116          if(y > 0 and x < m_Area.get_height())
117              yc = y;
118      }
119      else
120      {
121          xc = m_Area.get_width()/2 ;
122          yc = m_Area.get_height()/2 ;
123      }
124      m_Area.queue_draw();
125  }
```

```
A => mouse move detected and processed
B => mouse move detected and processed
C => mouse move detected and processed
D => mouse move detected and processed
E => mouse move detected and processed
F => mouse move detected and processed
G => mouse move detected and processed
H => mouse move detected and processed
mouse left clic detected and processed
x = 73   y = 96
mouse right clic detected and processed
I => mouse move detected and processed
J => mouse move detected and processed
K => mouse move detected and processed
```

lancer l'exécutable pour constater l'effet sur le dessin

Rappels et Résumé

- **Comme pour les autres éléments d'une interface graphique, la clef de leur fonctionnement est l'usage d'attributs pour mémoriser un *état désiré* de l'application**
- **ne pas oublier d'appeler `queue_draw()` si le changement désiré à un impact visuel.**
- **les événements de mouvement de la souris sonts produit seulement si un mouvement minimum est effectué.**