Créer son propre PAINT de Kitone

Table des matières

Créer son propre PAINT.

Maintenant que l'on connaît GTK+ il va falloir en faire quelque chose...mais quoi ? Gimp −13.0 voila notre défi ;) Bien sûr rien de bien compliqué, une simple fenêtre avec une barre d'outil, une barre d'état, un espace de travail (image), puis à sa droite quelques outils comme la taille du pinceau, les couleurs, etc...

Pourquoi un logiciel de dessin ?
Cela va nous permettre d'utiliser quelques fonctions GDK ainsi que GtkDrawingArea et tous les signaux comme le mouvement et clic de sourie avec un peu de DND.

Note : je ne vais pas commenter tout le code, pour plus d'info www.gtk−fr.org ! Ne chercher pas le code ou l'application parfaite, je fais simple, rapide, efficace ! L'application ne sera pas fini, pas de Ouvrir ou Sauvegarder , etc...juste le minimum.

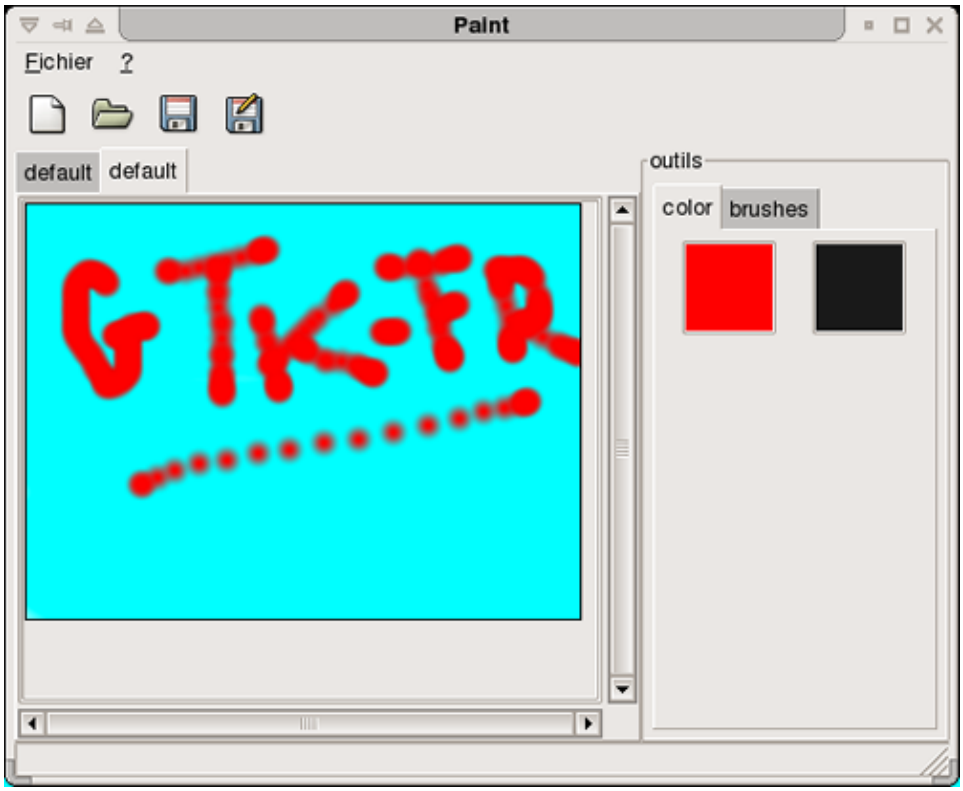
Qu'est ce que l'on veut ?

- Pouvoir utiliser les 2 boutons de la sourie avec pour chaque bouton une couleur différente.
- Pouvoir ouvrir plusieurs documents en même temps.
- Pouvoir changer la taille et le style du pinceau.
- Avoir l'application dans une seule fenêtre, voir quelques fenêtres sous forme d'outil (pour utiliser les Dook).
- Aussi pouvoir dessiner si possible :)

C'est déjà pas mal...

Pour la gestion des documents, nous allons utiliser les "tabs" (GtkNoteBook), cela permet d'avoir une seule interface pour plusieurs documents afin qu'ils n'occupent pas tout votre espace. La même solution sera adoptée pour les outils, cela ne prend pas de place et pour chaque nouveaux outils (ou groupe) il suffit d'ajouter une page dans notre "notebook".

Voila une image qui montre l'application finale :



Nous allons ensuite utiliser des structures pour notre application, ceci permet une plus grand souplesse et un grand nombre de variable à porté de main :

```
struct paint_app
{
```

```
   GtkWidget *window; /* fenêtre principale */
   GtkWidget *vbox; /* box principale directement dans la fenêtre */
   GtkWidget *menu_bar; /* notre barre pour le menu */
   GtkWidget *toolbar; /* barre d'outil */
   GtkWidget *hbox; /* box contenant nos documents + outils */
   GtkWidget *notebook_doc; /* notre NoteBook contenant les documents */
   GtkWidget *tools; /* frame pour nos outils (contient le notebook_doc) */
   GtkWidget *notebook_tools; /* NoteBook qui contient nos outils */
   GtkWidget *statusbar; /* barre d'état */

   GList *list_documents; /* liste des documents */

   paint_brush brush; /* notre pinceau */
};
```

Structure pour notre pinceau :

```
struct paint_brush
{
   BlendingMode blending_mode; /* méthode de melange entre le fond et la nouvelle couleur */
   gfloat opacity; /* opacité de notre pinceau */
   guint size; /* rayon de notre pinceau */
   gfloat color_a[3]; /* couleur du bouton 1 */
   gfloat color_b[3]; /* couleur du bouton 2 */
};
```

Création de la fenêtre

Rien de bien compliqué, une fenêtre du type TOPLEVEL fera l'affaire. Nous allons diviser cette fenêtre en 3 groupes :

- le menu (Fichier, Edition, About, etc...) ainsi que la barre d'outils
- l'espace de travail ainsi que nos outils (à droite)
- la barre d'état

Pour faire cela nous allons ajouter une GtkVBox pour contenir cela, le 1er et 3eme groupe ne doit occuper que l'espace minimum alors que notre espace de travail doit occuper le maximum d'espace et doit être redimensionnable.

Nous allons configurer cela grâce à la fonction gtk_box_pack_start :

```
/* pour le menu */
gtk_box_pack_start(GTK_BOX(app->vbox), app->menu_bar, FALSE, TRUE, 0);
/* pour la barre d'outil */
app->toolbar=gtk_toolbar_new();
gtk_box_pack_start(GTK_BOX(app->vbox), app->toolbar, FALSE, TRUE, 0);
/* pour l'espace de travail, HBox qui va contenir l'image+outils */
app->hbox=gtk_hbox_new(FALSE, 0);
gtk_box_pack_start(GTK_BOX(app->vbox), app->hbox, TRUE, TRUE, 0);
```

Création du menu

Nous allons utiliser la méthode rapide :

(note : les signaux ne sont pas encore connectés et le menu ne sert à rien ! c'est pour la forme)

```
static GtkItemFactoryEntry MenuItem[] = {
   { "/_Fichier", NULL, NULL, 0, "<Branch>" },
   { "/Fichier/_Nouveau", NULL, NULL, 0, "<StockItem>", GTK_STOCK_NEW },
   { "/Fichier/_Ouvrir", NULL, NULL, 0, "<StockItem>", GTK_STOCK_OPEN },
   { "/Fichier/Enregi_strer", "<ctrl>S", NULL, 0, "<StockItem>", GTK_STOCK_SAVE },
   { "/Fichier/_Fermer", "<ctrl>F", NULL, 0, "<StockItem>", GTK_STOCK_CLOSE },
   { "/Fichier/Sep1", NULL, NULL, 0, "<Separator>" },
   { "/Fichier/_Quitter", NULL,NULL, 1, "<StockItem>", GTK_STOCK_QUIT},
   { "/_?", NULL, NULL, 0, "<Branch>" },
   { "/?/_A propos de...", "<CTRL>A", NULL, 1, "<StockItem>", GTK_STOCK_HELP}
};

/* Nombre d'élément du menu */
static gint iNbMenuItem = sizeof(MenuItem) / sizeof(MenuItem[0]);

/* création de la barre d'outil */
void paint_create_menu_bar(paint_app *paint)
{
   GtkItemFactory *pItemFactory;
   GtkAccelGroup *pAccel;

   /* Création de la table d'accélération */
   pAccel = gtk_accel_group_new ();
   /* Création du menu */
   pItemFactory = gtk_item_factory_new(GTK_TYPE_MENU_BAR, "<main>", pAccel);
   /* Récupération des éléments du menu */
   gtk_item_factory_create_items(pItemFactory, iNbMenuItem, MenuItem, (GtkWidget*)paint->window);
   /* Récupération du widget pour l'affichage du menu */
   paint->menu_bar = gtk_item_factory_get_widget(pItemFactory, "<main>");
   /* Association des raccourcis avec la fenêtre */
   gtk_window_add_accel_group(GTK_WINDOW(paint->window), pAccel);
   gtk_widget_show_all(paint->menu_bar);
}
```

Création de l'espace de travail

Alors sur quoi allons-nous dessiner ? Sur une GtkDrawingArea, c'est le moyen le plus simple et rapide pour faire cela. Bien sûr pour avoir une gestion efficace de nos documents et une souplesse d'utilisation nous n'allons pas directement dessiner sur la drawing_area mais nous allons passer par un "buffer". Cela va permettre plusieurs choses, la possibilité future d'utiliser les calques (si j'ai le courage de le faire), la possibilité d'appliquer des effets (encore mon courage), etc...

Alors à quoi faut-il penser avant de créer notre DrawingArea ? A la taille de l'image, celle-ci ne doit pas changer si l'application subit un redimensionnement, mais l'application doit pouvoir contenir plusieurs

documents de taille différente, donc pour permettre cela, l'image doit être contenue dans des barres de défilement.

Chaque Document va avoir sa propre structure :

```
struct paint_document
{
   guchar *name;
   GtkWidget *drawing_area;
   guchar *buffer;
   guint width;
   guint height;
};
```

- drawing_area représente, comme son nom l'indique, la surface où l'on va rendre l'image à l'écran.
- buffer représente l'image, un tableau : width*height*3, les couleurs sont comprises entre 0 et 255.
- width et height représentent la taille de l'image.

La première chose à faire est de créer nos deux parties (image+tools) comme il est dit plus haut, dans les 2 cas nous allons utiliser GtkNoteBook :

Voila le code qui permet de créer notre fenêtre avec tous les éléments en place :

```
void paint_create(paint_app *app)
{
   ptr_paint_app=app;

   app−>list_documents=NULL;

   /* init brush */
   app−>brush.blending_mode=NORMAL;
   app−>brush.opacity=1.0f;
   app−>brush.size=10;

   app−>brush.color_a[0]=app−>brush.color_a[1]=app−>brush.color_a[2]=0.6f;
   app−>brush.color_b[0]=app−>brush.color_b[1]=app−>brush.color_b[2]=0.1f;

   /* win */
   app−>window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(app−>window), "Paint");
   gtk_window_set_default_size(GTK_WINDOW(app−>window), 400, 300);
   g_signal_connect(G_OBJECT(app−>window), "destroy", G_CALLBACK(gtk_main_quit), NULL);

   /* hbox qui contient notre interface (dans l'ordre toolbar+espace de travail+barre d'état) */
   app−>vbox=gtk_vbox_new(FALSE, 0);
   gtk_container_add(GTK_CONTAINER(app−>window), app−>vbox);

   /* menu bar */
   pain_create_menu_bar(app);
   gtk_box_pack_start(GTK_BOX(app−>vbox), app−>menu_bar, FALSE, TRUE, 0);
```

```
    /* toolbar */
    paint_create_toolbar(app);

    gtk_box_pack_start(GTK_BOX(app->vbox), app->toolbar, FALSE, TRUE, 0);

    /* HBOX */
    app->hbox=gtk_hbox_new(FALSE, 0);
    gtk_box_pack_start(GTK_BOX(app->vbox), app->hbox, TRUE, TRUE, 0);

    /* notebook c'est là dedans que sont nos documents */
    app->notebook_doc=gtk_notebook_new();

    /* nous voulons que le notebook occupe toute la place car c'est ici que l'on dessine */
    gtk_box_pack_start(GTK_BOX(app->hbox), app->notebook_doc, TRUE, TRUE, 0);

    /* la frame tools, qui va contenir le notebook tools créée juste en dessous par la fonction paint_create_tools
*/
    app->tools=gtk_frame_new("outils");
    gtk_frame_set_shadow_type(GTK_FRAME(app->tools), GTK_SHADOW_IN);
    gtk_box_pack_end(GTK_BOX(app->hbox), app->tools, FALSE, TRUE, 0);
    /* création des outils */
    paint_create_tools(app);

    /* création de la barre d'état */
    app->statusbar=gtk_statusbar_new();
    gtk_box_pack_end(GTK_BOX(app->vbox), app->statusbar, FALSE, TRUE, 0);

    /* création d'un document vide */
    paint_add_document("default", 320, 240);

    /* show all */
    gtk_widget_show_all(app->window);
}
```

Création des documents

Il y a 3 étapes dans la vie d'un document, la création, l'utilisation, la destruction.

Création :

En plus de créer notre document, nous devons créer le "buffer" pour dessiner dedans, celui-ci dépend de la taille du document. Pour plus de souplesse, nous allons laisser à l'utilisateur le choix de la taille et du nom du document à créer, pour cela nous allons connecter le bouton "Nouveau" de la barre d'outil et l'item du menu fichier à un "callback" qui va nous ouvrir une fenêtre du type dialogue en modal. Celle-ci va nous permettre d'entrer la taille du document (hauteur, largeur) et le nom du document (pour les tabs).

Le code qui ouvre une fenêtre avec une GtkEntry et 2 GtkSpinButton :

```
void paint_create_new_document(GtkWidget *w, gpointer data)
{
   GtkDialog *dialog;
   GtkTable *table;
   GtkWidget *label;
   GtkWidget *entry_name;
   GtkWidget *spin_width;
   GtkWidget *spin_height;

   dialog=GTK_DIALOG(gtk_dialog_new_with_buttons("nouveau document",
      GTK_WINDOW(paint_get_instance()->window),
      GTK_DIALOG_MODAL,
      GTK_STOCK_OK, GTK_RESPONSE_OK,
      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
      NULL));


   table=GTK_TABLE( gtk_table_new(3,3, TRUE) );
   gtk_container_set_border_width(GTK_CONTAINER(table), 5);
   gtk_box_pack_start(GTK_BOX(dialog->vbox), GTK_WIDGET(table), FALSE, FALSE, 0);

   /* nom du document */
   label=gtk_label_new("name");
   gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);
   entry_name = gtk_entry_new();
   gtk_entry_set_text(GTK_ENTRY(entry_name), "default");

   gtk_table_attach_defaults(table, label, 0,1,0,1);
   gtk_table_attach_defaults(table, entry_name, 1,3,0,1);

   /* taille du document width */
   label=gtk_label_new("width");
   gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);

   spin_width = gtk_spin_button_new_with_range(10, 800,1);
   gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_width), 320);
   gtk_table_attach_defaults(table, label, 0,1,1,2);
   gtk_table_attach_defaults(table, spin_width, 1,3,1,2);
   /* taille du document height */
```

```
   label=gtk_label_new("height");
   gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);

   spin_height = gtk_spin_button_new_with_range(10, 800,1);
   gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_height), 240);
   gtk_table_attach_defaults(table, label, 0,1,2,3);
   gtk_table_attach_defaults(table, spin_height, 1,3,2,3);

   gtk_widget_show_all(GTK_WIDGET(dialog));

   switch(gtk_dialog_run(dialog))
   {
      case GTK_RESPONSE_OK:
         guint w=(guint)gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_width));
         guint h=(guint)gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_height));
         const gchar *label=gtk_entry_get_text(GTK_ENTRY(entry_name));
         paint_add_document(label, w,h);
         break;
      case GTK_RESPONSE_CANCEL:
      default:
         break;
   }

   gtk_widget_destroy(GTK_WIDGET(dialog));
}
```

Rien de bien compliqué, on crée une boite de dialogue avec quelques widgets, puis si la réponse est OK on récupère nos données pour créer notre nouveau document grâce à la fonction paint_add_document.

Utilisation :

Alors c'est ici qu'il faut créer notre widget pour qu'il soit utilisable, pouvoir récupérer les mouvements de la sourie, les boutons pressés, s'afficher, etc...

```
paint_document* paint_add_document(const gchar *label, guint width, guint height)
{
   paint_app *paint=paint_get_instance();

   paint_document *doc;
   GtkWidget *frame;
   GtkWidget *scrollbar;

   g_return_val_if_fail(paint, NULL);
   /* création du document */
   doc=(paint_document*)g_malloc(sizeof(paint_document));
   g_return_val_if_fail(doc, NULL);

   /* on crée le widget */
   doc->drawing_area=gtk_drawing_area_new();

   /* on remplit notre structure */
```

```
    doc->width=width;
    doc->height=height;
    /* alloue le buffer */
    doc->buffer=(guchar*)g_malloc(sizeof(guchar)*doc->width*doc->height*3);

    /* remplit le buffer avec la couleur blanche */
    for(guint x=0; x<doc->width; ++x)
       for(guint y=0; y<doc->height; ++y)
       {
          doc->buffer[(y*doc->width+x)*3]=255;
          doc->buffer[(y*doc->width+x)*3+1]=255;
          doc->buffer[(y*doc->width+x)*3+2]=255;
       }

    /* donne la taille de notre image */
    gtk_drawing_area_size(GTK_DRAWING_AREA(doc->drawing_area),doc->width, doc->height);

    /* donne les possibilités à notre widget */
    gtk_widget_set_events(doc->drawing_area, GDK_EXPOSURE_MASK
       | GDK_LEAVE_NOTIFY_MASK | GDK_BUTTON_PRESS_MASK
       | GDK_POINTER_MOTION_MASK );

    /* configure les signaux */
    g_signal_connect(G_OBJECT(doc->drawing_area), "realize", G_CALLBACK(paint_document_realize),
(gpointer)doc);
    g_signal_connect(G_OBJECT(doc->drawing_area), "expose_event",
G_CALLBACK(paint_document_expose), (gpointer)doc);
    g_signal_connect(G_OBJECT(doc->drawing_area), "button_press_event",
G_CALLBACK(paint_document_button_press), (gpointer)doc);
    g_signal_connect(G_OBJECT(doc->drawing_area), "button_release_event",
G_CALLBACK(paint_document_button_release), (gpointer)doc);
    g_signal_connect(G_OBJECT(doc->drawing_area), "motion_notify_event",
G_CALLBACK(paint_document_motion_notify), (gpointer)doc);

    /* vraiment utile ?? */
    gtk_widget_set_app_paintable(doc->drawing_area, TRUE);

    /* ajoute le document à la liste */
    paint->list_documents = g_list_append(paint->list_documents, doc);

    /* scrool qui va contenir une frame, qui elle même va contenir notre drawing_area */
    scrollbar = gtk_scrolled_window_new(NULL, NULL);
    gtk_notebook_append_page(GTK_NOTEBOOK(paint->notebook_doc), scrollbar, gtk_label_new(label));

    /* frame qui va contenir notre drawing_area */
    frame=gtk_frame_new(NULL);
    gtk_frame_set_shadow_type(GTK_FRAME(frame), GTK_SHADOW_IN);
    gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrollbar), frame);
    /* on ajoute notre drawing_area à la frame */
    gtk_container_add(GTK_CONTAINER(frame), doc->drawing_area);
```

```
    gtk_widget_show_all(scrollbar);

    return doc;
}
```

Alors je vais expliquer quelques points importants de ce code pour le rendre plus clair :

Par défaut, une drawing_area n'a aucun signal, donc si vous connectez par exemple le signal "button_press_event", vous n'aurez jamais ce signal d'appeler si vous cliquez sur la surface, car votre widget ne possède pas cette capacité par défaut. Il faut pour cela lui donner un ensemble "d'event" que l'on souhaite pouvoir recevoir. Pour faire cela il suffit d'utiliser gtk_widget_set_events ou gtk_widget_add_events. (ceci est valable pour n'importe quel widget). Dans notre cas, nous avons besoin d'avoir la possibilité de dessiner (expose), de savoir si un bouton est pressé ou relâché (press, release) et de savoir quand la sourie bouge (motion).

```
gtk_widget_set_events(doc->drawing_area, GDK_EXPOSURE_MASK
    | GDK_BUTTON_RELEASE_MASK | GDK_BUTTON_PRESS_MASK
    | GDK_POINTER_MOTION_MASK );
```

Ensuite il faut récupérer ces signaux, pour cela on connecte notre drawing_area à chaque signal :

- "realize" est le moment où le widget est vraiment créé, c'est à ce moment que nous devons lui fournir notre nouveau cursor. (pour donner un GdkCursor il faut que widget->window existe, si le widget n'est pas réalisé, alors il est égal à NULL donc pas possible de créer votre cursor).
- "expose_event" est la fonction qui va être appelée si le widget doit être dessiné (en totalité ou en parti)
- "button_press_event" est le signal envoyé si on appuie sur un bouton de la sourie.
- "button_release_event" quand on relâche un bouton de la sourie.
- "motion_notify_event" quand la sourie bouge.

Dans notre cas nous n'avons pas besoin de récupérer l'événement "configure_event" qui intervient quand le widget change de taille, puisque nos documents ont une taille fixe donc impossible à redimensionner. N'oublions pas qu'ils sont contenus dans des "scrollbar".

```
g_signal_connect(G_OBJECT(doc->drawing_area), "realize", G_CALLBACK(paint_document_realize),
(gpointer)doc);
g_signal_connect(G_OBJECT(doc->drawing_area), "expose_event",
G_CALLBACK(paint_document_expose), (gpointer)doc);
g_signal_connect(G_OBJECT(doc->drawing_area), "button_press_event",
G_CALLBACK(paint_document_button_press), (gpointer)doc);
g_signal_connect(G_OBJECT(doc->drawing_area), "button_release_event",
G_CALLBACK(paint_document_button_release), (gpointer)doc);
g_signal_connect(G_OBJECT(doc->drawing_area), "motion_notify_event",
G_CALLBACK(paint_document_motion_notify), (gpointer)doc);
```

Pour rendre la gestion plus simple nous envoyons un pointeur sur notre document avec le signal.

Destruction :

Vraiment rien de spécial, il suffit d'appeler cette fonction quand on quitte notre application, elle s'occupe simplement d'effacer les buffers alloués à chaque document et d'effacer la liste des documents :

```
void paint_delete(paint_app *paint)
{
   if(paint->list_documents!=NULL)
   {
      GList *temp;
      temp = g_list_first(paint->list_documents);

      while(temp)
      {
         paint_document *doc;

         doc=(paint_document*)temp->data;
         if(!doc->buffer)
            g_free(doc->buffer);

         g_free(temp->data);
         temp = g_list_next(temp);
      }

      g_list_free(paint->list_documents);
      paint->list_documents=NULL;
   }
   ptr_paint_app=NULL;
}
```

Comment rendre à l'écran notre image ?

Comme je vous l'ai dit au début, nous allons passer par un tampon pour dessiner, donc pour le rendre à l'écran il faut que GDK le dessine. Voila la fonction "expose" qui s'occupe simplement de dessiner ce qu'il y a dans le tampon. On fait très simple, on dessine toujours tout le tampon (de 0,0 a width,height).

```
gboolean paint_document_expose(GtkWidget *drawing, GdkEventExpose *e, gpointer data)
{
   g_return_val_if_fail(data, FALSE);

   paint_document *doc;
   doc=(paint_document*)data;

   /* notre buffer est il créé ? */
   if(!doc->buffer)
      return TRUE;

   gdk_draw_rgb_image (drawing->window, drawing->style->fg_gc[GTK_WIDGET_STATE(drawing)],
      0,0,
      doc->width, doc->height,
      GDK_RGB_DITHER_MAX,
      doc->buffer,
      doc->width*3);

   gdk_draw_rectangle(drawing->window, drawing->style->black_gc, FALSE, 0,0,doc->width,
```

```
doc->height);
   return TRUE;
}
```

Petite info :

Dans tout ce qu'il y a à voir avec les "callback" GDK il faut retourner une gboolean, car GDK est plus bas niveau que GTK. Ceci nous permet d'intercepter des signaux et on a le choix entre les laisser se propager ou les stopper. Si on retourne TRUE alors on dit à GDK que nous avons traité le signal, il en conclu qu'il n'y a plus besoin de le propager, FALSE fait l'inverse.
Cela peut être utile dans quelques cas, si par exemple vous désirez intercepter le signal d'un widget (ex : un GtkButton) et que vous désirez effectuer quelque chose avant que le signal ne lui soit émis, ceci permet d'effectuer l'action puis selon notre choix (retourne TRUE ou FALSE) on redonne la main au widget.

Dans un premier temps, on regarde si notre document a un buffer créé (on sait jamais, mais normalement il doit l'être), puis on passe par GDK pour dessiner le contenu de notre buffer et enfin nous dessinons un rectangle qui représente les contours du document. (Regarder la documentation GDK pour voir comment fonctionne ces deux fonctions, c'est très simple)

Comment dessiner ?

C'est bien beau, mais maintenant que l'on a une interface et que l'on peut rendre notre buffer à l'écran, il va falloir modifier ce buffer ! Ceci n'a presque rien à voir avec GDK ou GTK.

La première étape va être de récupérer les clics de l'utilisateur, selon que l'on clique à gauche ou à droite nous allons utiliser une couleur différente (on peut choisir la couleur de chaque bouton dans la page "color" de nos outils). Donc à chaque clic on doit dessiner, mais si l'utilisateur garde le bouton appuyé et qu'il bouge la sourie on doit encore dessiner ! Pour cela on va créer une fonction apply_brush , on va lui fournir le document en cours, et un point (x,y) cette fonction va s'occuper de dessiner dans le buffer.

La sourie :

```
/* PRESS */
gboolean paint_document_button_press(GtkWidget *w, GdkEventButton *e, gpointer data)
{
   gint x;
   gint y;
   GdkModifierType state;

   paint_document *doc=(paint_document*)data;

   if(e->button == 1 && doc->buffer != NULL)
   {
      gdk_window_get_pointer (w->window, &x, &y, &state);
      apply_brush(doc, x,y, paint_get_instance()->brush.color_a);
      gtk_widget_queue_draw(w);
   }
   if(e->button == 3 && doc->buffer != NULL)
   {
      gdk_window_get_pointer (w->window, &x, &y, &state);
```

```
      apply_brush(doc, x,y, paint_get_instance()->brush.color_b);
      gtk_widget_queue_draw(w);
   }
   return TRUE;
}

/* RELEASE */
gboolean paint_document_button_release(GtkWidget *w, GdkEventButton *e, gpointer data)
{
   /* pour l'instant vide */
   return TRUE;
}

/* MOTION */
gboolean paint_document_motion_notify(GtkWidget *w, GdkEventMotion *e, gpointer data)
{
   gint x;
   gint y;
   GdkModifierType state;

   paint_document *doc=(paint_document*)data;

   if(e->is_hint)
      gdk_window_get_pointer(w->window, &x, &y, &state);
   else
   {
      x=e->x;
      y=e->y;
      state=(GdkModifierType)e->state;
   }

   if((state & GDK_BUTTON1_MASK) && doc->buffer != NULL)
   {
      apply_brush(doc, x,y, paint_get_instance()->brush.color_a);
      gtk_widget_queue_draw(w);
   }
   else if((state & GDK_BUTTON3_MASK) && doc->buffer != NULL)
   {
      apply_brush(doc, x,y, paint_get_instance()->brush.color_b);
      gtk_widget_queue_draw(w);
   }
   return TRUE;
}
```

Rien de bien compliqué, si l'utilisateur appuie sur le bouton de sa sourie, alors on récupère les coordonnées (par rapport au widget grâce à la fonction gdk_window_get_pointer) puis on envoie ces coordonnées à notre fonction apply_brush avec la couleur (color_a ou color_b selon le type de bouton) une fois dessinée dans le buffer, on utilise gtk_widget_queue_draw pour demander à GTK de redessiner notre surface (drawing_area).

apply_brush :

Comment dessiner ?? Nous n'allons bien sûr pas dessiner avec un rectangle ou un carré ! Trop brutal ! nous allons passer par un cercle avec un petit dégradé. Quand on appelle cette fonction, on ne doit pas dessiner que sur le pixel (x,y) mais aussi autour car on peut choisir la taille du pinceau. En plus de pouvoir changer la taille du pinceau, nous allons aussi permettre de choisir la manière de mélanger les couleurs.

Pour le mélange des couleurs une fonction :

```
void blending(gfloat *res, gfloat *A, gfloat *B, paint_brush *brush, gfloat r)
{
   guint i;

   switch(brush->blending_mode)
   {
      case NORMAL:
         for(i=0; i<3; ++i) res[i]=A[i]+(B[i]-A[i])*brush->opacity*r;
         break;
      case ADD:
         for(i=0; i<3; ++i) res[i]=(B[i]*brush->opacity*r)+A[i];
         break;
      case SUB:
         for(i=0; i<3; ++i) res[i]=A[i]-B[i]*brush->opacity*r;
         break;
      /* etc */
```

Regarder les sources pour plus d'info (disponibles en téléchargement).

Puis la fonction apply_brush :

```
/* applique un pinceau au coord (x,y) (+taille de pinceau) */
void apply_brush(paint_document *doc, gint x, gint y, gfloat *color)
{
   gfloat tmp[3];
   gfloat dest[3];

   guint size=paint_get_instance()->brush.size;

   gint startx;
   gint starty;
   gint endx;
   gint endy;
   gint i,j;

   /* on déborde ? */
   if(x<0 || x>doc->width) return;
   if(y<0 || y>doc->height) return;

   /* on clamp */
   startx=x-size;
   clamp(startx, 0, doc->width);
```

```
    endx=x+size;
    clamp(endx, 0, doc->width);

    starty=y-size;
    clamp(starty, 0, doc->height);
    endy=y+size;
    clamp(endy, 0, doc->height);

    /* on dessine */
    for(j = starty; j < endy; ++j)
    {
        for(i = startx; i < endx; ++i)
        {
            doc_get_color(doc, i,j, tmp);

            gfloat sx=x-i;
            gfloat sy=y-j;
            gfloat d=sqrtf(sx*sx + sy*sy);
            gfloat r=CubicSpline(0,size, d);

            blending(dest, tmp, color, &paint_get_instance()->brush, 1-r);
            doc_set_color(doc, i,j, dest);
        }
    }
}
```

Elle est très simple, selon les coordonnées (x,y) on dessine autour de ce point avec une intensité différente selon qu'on soit au milieu (donc x,y) ou au bord du pinceau. Pour dessiner, on récupère(dans le buffer) dans un premier temps la couleur du pixel (i,j), puis on cherche l'intensité du pinceau (r) et enfin on mélange les couleurs grâce à la fonction blending, et pour finir on remplit le pixel (toujours dans le buffer) avec cette nouvelle couleur.

Note : nous travaillons toujours avec des couleurs comprises entre [0,1] pour plus de facilité, on les convertit ensuite en RGB [0,255].

Conclusion :

Hum....bien sûr ce n'est pas une application finie, par exemple la barre d'état...ne sert à rien, ainsi que les menus (sauf ficher−>nouveau), idem pour la barre d'outil, seul nouveau est connecté, le but était juste de mettre en place quelques éléments, à vous de combler les trous... J'ajouterais quelques morceaux de code et autres modifications plus tard comme par exemple pouvoir lire une image au format png ou jpeg, pouvoir sauvegarder l'image, ajouter d'autre pinceau, etc...si le courage me le permet.

Bon GTK+

Kitone [Kitone] (@) [wanadoo.fr]
www.gtk−fr.org

Annexe 1 − Code Source

brushes.h

```
#ifndef _brushes_h
#define _brushes_h

#include "paint.h"

void paint_create_tools_brushes();

#endif //_brushes_h
```

brushes.cc

```
#include "brushes.h"

void brushes_blending_changed(GtkWidget *w, gpointer data)
{
   paint_app *app=paint_get_instance();

   guint num=gtk_option_menu_get_history(GTK_OPTION_MENU(w));
   app−>brush.blending_mode=(BlendingMode)num;
}

void brushes_opacity_changed(GtkWidget *w, gpointer data)
{
   paint_app *app=paint_get_instance();

   gfloat val=gtk_spin_button_get_value_as_float(GTK_SPIN_BUTTON(w));
   app−>brush.opacity=val;
}

void brushes_size_changed(GtkWidget *w, gpointer data)
{
   paint_app *app=paint_get_instance();

   guint val=(guint)gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(w));
   app−>brush.size=val;
}

void paint_create_tools_brushes()
{
   GtkWidget *table;
   GtkWidget *option_menu;
   GtkWidget *menu;
   GtkWidget *item;
   GtkWidget *spin_opacity;
   GtkWidget *spin_size;
```

```
    paint_app *paint=paint_get_instance();
    /* brushes */
    table=gtk_table_new(3,2, TRUE);
    gtk_container_set_border_width(GTK_CONTAINER(table), 5);
    gtk_notebook_append_page(GTK_NOTEBOOK(paint->notebook_tools), table,
gtk_label_new("brushes"));
    // blending
    menu=gtk_menu_new();
    item=gtk_menu_item_new_with_label("normal"); gtk_menu_shell_append(GTK_MENU_SHELL(menu),
item);
    item=gtk_menu_item_new_with_label("add"); gtk_menu_shell_append(GTK_MENU_SHELL(menu),
item);
    item=gtk_menu_item_new_with_label("sub"); gtk_menu_shell_append(GTK_MENU_SHELL(menu),
item);

    option_menu=gtk_option_menu_new();
    gtk_option_menu_set_menu(GTK_OPTION_MENU(option_menu), menu);
    gtk_table_attach(GTK_TABLE(table), gtk_label_new("blending"), 0,1,0,1,GTK_FILL,GTK_FILL,0,0);
    gtk_table_attach(GTK_TABLE(table), option_menu, 1,2,0,1,GTK_FILL,GTK_FILL,0,0);
    g_signal_connect(G_OBJECT(option_menu), "changed", G_CALLBACK(brushes_blending_changed),
NULL);
    // opacity
    spin_opacity=gtk_spin_button_new_with_range(0.0,1.0, 0.001);
    gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_opacity), paint->brush.opacity);
    gtk_table_attach(GTK_TABLE(table), gtk_label_new("opacity"), 0,1,1,2,GTK_FILL,GTK_FILL,0,0);
    gtk_table_attach(GTK_TABLE(table), spin_opacity, 1,2,1,2,GTK_FILL,GTK_FILL,0,0);
    g_signal_connect(G_OBJECT(spin_opacity), "value-changed",
G_CALLBACK(brushes_opacity_changed), NULL);
    // size
    spin_size=gtk_spin_button_new_with_range(1,100, 1);
    gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_size), paint->brush.size);
    gtk_table_attach(GTK_TABLE(table), gtk_label_new("size"), 0,1,2,3,GTK_FILL,GTK_FILL,0,0);
    gtk_table_attach(GTK_TABLE(table), spin_size, 1,2,2,3,GTK_FILL,GTK_FILL,0,0);
    g_signal_connect(G_OBJECT(spin_size), "value-changed", G_CALLBACK(brushes_size_changed),
NULL);

    gtk_widget_show_all(table);
}
```

color.h

```
#ifndef _color_h
#define _color_h

#include "paint.h"

void paint_create_tools_color();

#endif //_color_h
```

color.cc

```
#include "color.h"

void set_color(GtkWidget *w, gfloat *color)
{
   GdkColor c;
   c.red =(gushort)(color[0]*65535);
   c.green =(gushort)(color[1]*65535);
   c.blue =(gushort)(color[2]*65535);
   gtk_widget_modify_bg(w, GTK_STATE_NORMAL, &c);
}

gboolean color_changed(GtkWidget *w, GdkEvent *e, gpointer data)
{
   gfloat *color_brush=(gfloat*)data;

   GdkColor start=w−>style−>bg[GTK_STATE_NORMAL];
   GdkColor dest;
   GtkWidget *win=gtk_color_selection_dialog_new("color");
   GtkColorSelectionDialog*w_d=GTK_COLOR_SELECTION_DIALOG(win);
   gtk_color_selection_set_current_color(GTK_COLOR_SELECTION(w_d−>colorsel), &start);
   switch(gtk_dialog_run(GTK_DIALOG(win)))
   {
      case GTK_RESPONSE_OK:
         gtk_color_selection_get_current_color(GTK_COLOR_SELECTION(w_d−>colorsel), &dest);
         color_brush[0]=(gfloat)(((gfloat)dest.red)/65535);
         color_brush[1]=(gfloat)((gfloat)(dest.green)/65535);
         color_brush[2]=(gfloat)((gfloat)(dest.blue)/65535);
         gtk_widget_modify_bg(w, GTK_STATE_NORMAL, &dest);
         //signal_emit
         break;
      default:
         break;
   }
   gtk_widget_destroy(GTK_WIDGET(win));
   gtk_widget_queue_draw(w);
   return TRUE;
}

void paint_create_tools_color()
{
   paint_app *paint=paint_get_instance();

   GtkWidget *table;
   GtkWidget *frame;
   GtkWidget *drawing_area;

   /* brushes */
   table=gtk_table_new(2,2, TRUE);
```

```
    gtk_container_set_border_width(GTK_CONTAINER(table), 5);
    gtk_notebook_append_page(GTK_NOTEBOOK(paint->notebook_tools), table, gtk_label_new("color"));

    // color A
    frame=gtk_frame_new(NULL);
    gtk_frame_set_shadow_type(GTK_FRAME(frame), GTK_SHADOW_IN);
    drawing_area=gtk_drawing_area_new();
    gtk_drawing_area_size(GTK_DRAWING_AREA(drawing_area), 50,50);
    gtk_widget_add_events(drawing_area, GDK_BUTTON_PRESS_MASK);
    gtk_container_add(GTK_CONTAINER(frame), drawing_area);
    gtk_table_attach(GTK_TABLE(table), frame, 0,1,0,1,GTK_EXPAND,GTK_FILL,0,0);
    set_color(drawing_area, paint->brush.color_a);
    g_signal_connect(G_OBJECT(drawing_area), "button_press_event", G_CALLBACK(color_changed),
(gpointer)paint->brush.color_a);

    // color B
    frame=gtk_frame_new(NULL);
    gtk_frame_set_shadow_type(GTK_FRAME(frame), GTK_SHADOW_IN);
    drawing_area=gtk_drawing_area_new();
    gtk_drawing_area_size(GTK_DRAWING_AREA(drawing_area), 50,50);
    gtk_widget_add_events(drawing_area, GDK_BUTTON_PRESS_MASK);
    gtk_container_add(GTK_CONTAINER(frame), drawing_area);
    gtk_table_attach(GTK_TABLE(table), frame, 1,2,0,1,GTK_EXPAND,GTK_FILL,0,0);
    set_color(drawing_area, paint->brush.color_b);
    g_signal_connect(G_OBJECT(drawing_area), "button_press_event", G_CALLBACK(color_changed),
(gpointer)paint->brush.color_b);

    gtk_widget_show_all(table);
}
```

paint.h

```
#ifndef _PAINT__H
#define _PAINT__H

#include <gtk/gtk.h>

/**
* methode de melange entre les couleurs
**/

enum BlendingMode
{
    NORMAL=0,
    ADD,
    SUB,
    // MUL,
    // DIV,
    // ALPHA,
    LAST
```

```
};

/**
* structure d'un document
**/

struct paint_document
{
   gchar* name; /* nom du document */
   GtkWidget* drawing_area; /* notre widget qui rend notre image */
   guchar* buffer; /* le tampon pour dessiner */
   guint width; /* taille de l'image, en largeur */
   guint height; /* hauteur */
};

/**
* notre pinceau
**/

struct paint_brush
{
   BlendingMode blending_mode; /* methode de melange entre le fond et la nouvelle couleur */
   gfloat opacity; /* opaciter de notre pinceau */
   guint size; /* rayon de notre pinceau */
   gfloat color_a[3]; /* couleur du bouton 1 */
   gfloat color_b[3]; /* couleur du bouton 2 */
};

/**
* l'application principale
**/

struct paint_app
{
   GtkWidget *window; /* fenetre principale */
   GtkWidget *vbox; /* box principale directement dans la fenetre */
   GtkWidget *menu_bar; /* notre bar pour le menu */
   GtkWidget *toolbar; /* bar d'outil */
   GtkWidget *hbox; /* box contenant nos document + outils */
   GtkWidget *notebook_doc; /* notre NoteBook contenant les documents */
   GtkWidget *tools; /* frame pour nos outils (contient le notebook_doc) */
   GtkWidget *notebook_tools; /* NoteBook qui contient nos outils */
   GtkWidget *statusbar; /* bar d'etat */

   GList *list_documents; /* list des documents */

   paint_brush brush; /* notre pinceau */
};

/**
* création de notre application
```

```
**/
void paint_create();

/**
* nous retourne le pointeur sur notre application
**/
paint_app* paint_get_instance();

/**
* detruit l'application
**/
void paint_delete();

/**
* créatin d'un nouveau document
**/
paint_document* paint_add_document(const gchar *label, guint width, guint height);

#endif //_PAINT__H
```

paint.cc

```
#include "paint.h"
#include "brushes.h"
#include "color.h"
#include <math.h>

void clamp(gint x, gint min, gint max)
{
   if(x<min) x=min;
   else if(x>max) x=max;
}

static paint_app *ptr_paint_app=NULL;

/**
* CALLBACK
**/
gboolean paint_document_configure(GtkWidget *w, GdkEventConfigure *e, gpointer data);
void paint_document_realize(GtkWidget *w, gpointer data);
gboolean paint_document_expose(GtkWidget *w, GdkEventExpose *e, gpointer data);
gboolean paint_document_button_press(GtkWidget *w, GdkEventButton *e, gpointer data);
gboolean paint_document_button_release(GtkWidget *w, GdkEventButton *e, gpointer data);
gboolean paint_document_motion_notify(GtkWidget *w, GdkEventMotion *e, gpointer data);

void paint_create_new_document(GtkWidget *w, gpointer data);
void apply_brush_cursor(paint_document *doc);

static GtkItemFactoryEntry MenuItem[] = {
   { "/_Fichier", NULL, NULL, 0, "<Branch>" },
```

```
   { "/Fichier/_Nouveau", NULL, G_CALLBACK(paint_create_new_document), 0, "<StockItem>",
GTK_STOCK_NEW },
   { "/Fichier/_Ouvrir", NULL, NULL, 0, "<StockItem>", GTK_STOCK_OPEN },
   { "/Fichier/Enregi_strer", "<ctrl>S", NULL, 0, "<StockItem>", GTK_STOCK_SAVE },
   { "/Fichier/_Fermer", "<ctrl>F", NULL, 0, "<StockItem>", GTK_STOCK_CLOSE },
   { "/Fichier/Sep1", NULL, NULL, 0, "<Separator>" },
   { "/Fichier/_Quitter", NULL,G_CALLBACK(gtk_main_quit), 1, "<StockItem>", GTK_STOCK_QUIT},
   { "/_?", NULL, NULL, 0, "<Branch>" },
   { "/?/_A propos de...", "<CTRL>A", NULL, 1, "<StockItem>", GTK_STOCK_HELP}
};

/* Nombre d elements du menu */
static gint iNbMenuItem = sizeof(MenuItem) / sizeof(MenuItem[0]);

/**
* création de la barre d'outil
**/
void paint_create_menu_bar()
{
   paint_app *paint=paint_get_instance();

   GtkItemFactory *pItemFactory;
   GtkAccelGroup *pAccel;

   /* Creation de la table d acceleration */
   pAccel = gtk_accel_group_new ();
   /* Creation du menu */
   pItemFactory = gtk_item_factory_new(GTK_TYPE_MENU_BAR, "<main>", pAccel);
   /* Recuperation des elements du menu */
   gtk_item_factory_create_items(pItemFactory, iNbMenuItem, MenuItem, (GtkWidget*)paint->window);
   /* Recuperation du widget pour l affichage du menu */
   paint->menu_bar = gtk_item_factory_get_widget(pItemFactory, "<main>");
   /* Association des raccourcis avec la fenetre */
   gtk_window_add_accel_group(GTK_WINDOW(paint->window), pAccel);
gtk_widget_show_all(paint->menu_bar);
}

/**
* Create Tools
**/
void paint_create_tools()
{
   paint_app *paint=paint_get_instance();
   paint->notebook_tools=gtk_notebook_new();
   gtk_container_add(GTK_CONTAINER(paint->tools), paint->notebook_tools);
   gtk_container_set_border_width(GTK_CONTAINER(paint->notebook_tools), 5);

   paint_create_tools_color();
   paint_create_tools_brushes();

   gtk_widget_show_all(paint->notebook_tools);
```

```
}

/**
* TOOLBAR
**/
void paint_create_toolbar()
{
   paint_app *app=paint_get_instance();
   app->toolbar=gtk_toolbar_new();

   gtk_toolbar_insert_stock(GTK_TOOLBAR(app->toolbar),GTK_STOCK_NEW ,NULL, NULL,
G_CALLBACK(paint_create_new_document), NULL,-1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(app->toolbar),GTK_STOCK_OPEN ,NULL, NULL,
G_CALLBACK(NULL), NULL,-1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(app->toolbar),GTK_STOCK_SAVE ,NULL, NULL,
G_CALLBACK(NULL), NULL,-1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(app->toolbar),GTK_STOCK_SAVE_AS ,NULL, NULL,
G_CALLBACK(NULL), NULL,-1);
}

/**
* Dialog Create Doc
**/
void paint_create_new_document(GtkWidget *w, gpointer data)
{
   GtkDialog *dialog;
   GtkTable *table;
   GtkWidget *label;
   GtkWidget *entry_name;
   GtkWidget *spin_width;
   GtkWidget *spin_height;

   dialog=GTK_DIALOG(gtk_dialog_new_with_buttons("nouveau documents",
      GTK_WINDOW(paint_get_instance()->window),
      GTK_DIALOG_MODAL,
      GTK_STOCK_OK, GTK_RESPONSE_OK,
      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
      NULL));


   table=GTK_TABLE( gtk_table_new(3,3, TRUE) );
   gtk_container_set_border_width(GTK_CONTAINER(table), 5);
   gtk_box_pack_start(GTK_BOX(dialog->vbox), GTK_WIDGET(table), FALSE, FALSE, 0);

   /* nom du document */
   label=gtk_label_new("name");
   gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);
   entry_name = gtk_entry_new();
   gtk_entry_set_text(GTK_ENTRY(entry_name), "default");

   gtk_table_attach_defaults(table, label, 0,1,0,1);
```

```
    gtk_table_attach_defaults(table, entry_name, 1,3,0,1);

    /* taille du document width */
    label=gtk_label_new("width");
    gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);

    spin_width = gtk_spin_button_new_with_range(10, 800,1);
    gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_width), 320);
    gtk_table_attach_defaults(table, label, 0,1,1,2);
    gtk_table_attach_defaults(table, spin_width, 1,3,1,2);
    /* taille du document height */
    label=gtk_label_new("height");
    gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_RIGHT);

    spin_height = gtk_spin_button_new_with_range(10, 800,1);
    gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin_height), 240);
    gtk_table_attach_defaults(table, label, 0,1,2,3);
    gtk_table_attach_defaults(table, spin_height, 1,3,2,3);

    gtk_widget_show_all(GTK_WIDGET(dialog));

    switch(gtk_dialog_run(dialog))
    {
       case GTK_RESPONSE_OK:
       {
          guint w=(guint)gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_width));
          guint h=(guint)gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_height));
          const gchar *label=gtk_entry_get_text(GTK_ENTRY(entry_name));

          paint_add_document(label, w,h);
       }break;
    case GTK_RESPONSE_CANCEL:
    default:
       break;
    }
    gtk_widget_destroy(GTK_WIDGET(dialog));
}

paint_document* paint_add_document(const gchar *label, guint width, guint height)
{
    paint_app *paint=paint_get_instance();

    paint_document *doc;
    GtkWidget *frame;
    GtkWidget *scrollbar;

    g_return_val_if_fail(paint, NULL);
    /* création du document */
    doc=(paint_document*)g_malloc(sizeof(paint_document));
    g_return_val_if_fail(doc, NULL);
```

```
   /* on crée le widget */
   doc->drawing_area=gtk_drawing_area_new();

   /* on remplit notre structure */
   doc->width=width;
   doc->height=height;
   /* alloue le buffer */
   doc->buffer=(guchar*)g_malloc(sizeof(guchar)*doc->width*doc->height*3);

   /* remplit le buffer avec la couleur blanche */
   for(guint x=0; x<doc->width; ++x)
      for(guint y=0; y<doc->height; ++y)
      {
         doc->buffer[(y*doc->width+x)*3]=255;
         doc->buffer[(y*doc->width+x)*3+1]=255;
         doc->buffer[(y*doc->width+x)*3+2]=255;
      }

   /* donne la taille de notre image */
   gtk_drawing_area_size(GTK_DRAWING_AREA(doc->drawing_area),doc->width, doc->height);

   /* donne les possibilités à notre widget */
   gtk_widget_set_events(doc->drawing_area, GDK_EXPOSURE_MASK
      | GDK_LEAVE_NOTIFY_MASK | GDK_BUTTON_PRESS_MASK
      | GDK_POINTER_MOTION_MASK );

   /* configure les signaux */
   g_signal_connect(G_OBJECT(doc->drawing_area), "realize", G_CALLBACK(paint_document_realize),
(gpointer)doc);
   g_signal_connect(G_OBJECT(doc->drawing_area), "expose_event",
G_CALLBACK(paint_document_expose), (gpointer)doc);
   g_signal_connect(G_OBJECT(doc->drawing_area), "button_press_event",
G_CALLBACK(paint_document_button_press), (gpointer)doc);
   g_signal_connect(G_OBJECT(doc->drawing_area), "button_release_event",
G_CALLBACK(paint_document_button_release), (gpointer)doc);
   g_signal_connect(G_OBJECT(doc->drawing_area), "motion_notify_event",
G_CALLBACK(paint_document_motion_notify), (gpointer)doc);

   /* vraiment utile ?? */
   gtk_widget_set_app_paintable(doc->drawing_area, TRUE);

   /* ajoute le document à la liste */
   paint->list_documents = g_list_append(paint->list_documents, doc);

   /* scrool qui va contenir une frame, qui elle même va contenir notre drawing_area */
   scrollbar = gtk_scrolled_window_new(NULL, NULL);
   gtk_notebook_append_page(GTK_NOTEBOOK(paint->notebook_doc), scrollbar, gtk_label_new(label));

   /* frame qui va contenir notre drawing_area */
   frame=gtk_frame_new(NULL);
   gtk_frame_set_shadow_type(GTK_FRAME(frame), GTK_SHADOW_IN);
```

```
   gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrollbar), frame);
   /* on ajoute notre drawing_area à la frame */
   gtk_container_add(GTK_CONTAINER(frame), doc->drawing_area);

   gtk_widget_show_all(scrollbar);

   return doc;
}

gboolean quitte(GtkWidget *w, GdkEvent *e, gpointer data)
{
   GtkWidget* win;

   win = gtk_message_dialog_new (GTK_WINDOW(w),
      GTK_DIALOG_MODAL,
      GTK_MESSAGE_QUESTION,
      GTK_BUTTONS_YES_NO,
      "Etes-vous sur de vouloir\nquitter GtkPaint ?");

   switch(gtk_dialog_run(GTK_DIALOG(win)))
   {
      case GTK_RESPONSE_YES:
      {
         gtk_widget_destroy(win);
         gtk_main_quit();
         return FALSE;
      }break;
      case GTK_RESPONSE_NONE:
      case GTK_RESPONSE_NO:
         gtk_widget_destroy(win);
         break;
   }
   return TRUE;
}

void paint_create(paint_app *app)
{
   ptr_paint_app=app;

   app->list_documents=NULL;

   /* init brush */
   app->brush.blending_mode=NORMAL;
   app->brush.opacity=1.0f;
   app->brush.size=10;

   app->brush.color_a[0]=app->brush.color_a[1]=app->brush.color_a[2]=0.6f;
   app->brush.color_b[0]=app->brush.color_b[1]=app->brush.color_b[2]=0.1f;

   /* win */
   app->window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
    gtk_window_set_title(GTK_WINDOW(app->window), "Paint");
    gtk_window_set_default_size(GTK_WINDOW(app->window), 400, 300);
    g_signal_connect(G_OBJECT(app->window), "destroy", G_CALLBACK(gtk_main_quit), NULL);

    /* hbox qui contient notre interface (dans l'ordre toolbar+espace de travail+barre d'état) */
    app->vbox=gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(app->window), app->vbox);

    /* menu bar */
    pain_create_menu_bar(app);
    gtk_box_pack_start(GTK_BOX(app->vbox), app->menu_bar, FALSE, TRUE, 0);

    /* toolbar */
    paint_create_toolbar(app);

    gtk_box_pack_start(GTK_BOX(app->vbox), app->toolbar, FALSE, TRUE, 0);

    /* HBOX */
    app->hbox=gtk_hbox_new(FALSE, 0);
    gtk_box_pack_start(GTK_BOX(app->vbox), app->hbox, TRUE, TRUE, 0);

    /* notebook c'est là dedans que sont nos documents */
    app->notebook_doc=gtk_notebook_new();

    /* nous voulons que le notebook occupe toute la place car c'est ici que l'on dessine */
    gtk_box_pack_start(GTK_BOX(app->hbox), app->notebook_doc, TRUE, TRUE, 0);

    /* la frame tools, qui va contenir le notebook tools créée juste en dessous par la fonction paint_create_tools
*/
    app->tools=gtk_frame_new("outils");
    gtk_frame_set_shadow_type(GTK_FRAME(app->tools), GTK_SHADOW_IN);
    gtk_box_pack_end(GTK_BOX(app->hbox), app->tools, FALSE, TRUE, 0);
    /* création des outils */
    paint_create_tools(app);

    /* création de la barre d'état */
    app->statusbar=gtk_statusbar_new();
    gtk_box_pack_end(GTK_BOX(app->vbox), app->statusbar, FALSE, TRUE, 0);

    /* création d'un document vide */
    paint_add_document("default", 320, 240);

    /* show all */
    gtk_widget_show_all(app->window);
}

/**
* retourn l'instance
**/
paint_app* paint_get_instance()
{
```

```
   if(ptr_paint_app==NULL)
      paint_create();

   return ptr_paint_app;
}

/**
* free notre application ainsi que les doc et buffer
**/
void paint_delete(paint_app *paint)
{
   if(paint->list_documents!=NULL)
   {
      GList *temp;
      temp = g_list_first(paint->list_documents);

      while(temp)
      {
         paint_document *doc;

         doc=(paint_document*)temp->data;
         if(!doc->buffer)
            g_free(doc->buffer);

         g_free(temp->data);
         temp = g_list_next(temp);
      }

      g_list_free(paint->list_documents);
      paint->list_documents=NULL;
   }
   ptr_paint_app=NULL;
}

/* REALIZE */
void paint_document_realize(GtkWidget *w, gpointer data)
{
   g_return_if_fail(data);

   GdkCursor *cursor=NULL;
   paint_document *doc;

   g_return_if_fail(data);
   doc=(paint_document*)data;

   cursor=gdk_cursor_new(GDK_PLUS);
   gdk_window_set_cursor(doc->drawing_area->window, cursor);
   gdk_cursor_unref(cursor);
}

/* EXPOSE */
```

```
gboolean paint_document_expose(GtkWidget *drawing, GdkEventExpose *e, gpointer data)
{
   g_return_val_if_fail(data, FALSE);

   paint_document *doc;

   doc=(paint_document*)data;
   /* notre buffer est il crée ? */
   if(!doc->buffer)
      return TRUE;

   gdk_draw_rgb_image (drawing->window, drawing->style->fg_gc[GTK_WIDGET_STATE(drawing)],
      0,0,
      doc->width, doc->height,
      GDK_RGB_DITHER_MAX,
      doc->buffer,
      doc->width*3);

   gdk_draw_rectangle(drawing->window, drawing->style->black_gc, FALSE, 0,0,doc->width,
doc->height);
   return TRUE;
}

/* DRAW */
gboolean doc_get_color(paint_document *doc, gint x, gint y, gfloat *color)
{
   if(x<0 || x>doc->width)
      return 0;
   if(y<0 || y>doc->height)
      return 0;

   color[0]=(gfloat)doc->buffer[(y*doc->width+x)*3]/255;
   color[1]=(gfloat)doc->buffer[(y*doc->width+x)*3+1]/255;
   color[2]=(gfloat)doc->buffer[(y*doc->width+x)*3+2]/255;
}

void doc_set_color(paint_document *doc, gint x, gint y, gfloat *color)
{
   if(x<0 || x>doc->width)
      return;
   if(y<0 || y>doc->height)
      return;

   doc->buffer[(y*doc->width+x)*3] =(guchar)(color[0]*255);
   doc->buffer[(y*doc->width+x)*3+1] =(guchar)(color[1]*255);
   doc->buffer[(y*doc->width+x)*3+2] =(guchar)(color[2]*255);
}

void mix_color(gfloat *dest,gfloat *a, gfloat *b, gfloat val)
{
   dest[0]=a[0]+(b[0]-a[0])*val;
```

```
    dest[1]=a[1]+(b[1]−a[1])*val;
    dest[2]=a[2]+(b[2]−a[2])*val;
}

void mix_alpha(gfloat *rgb, gfloat *rgb2, gfloat m)
{
    if(m>1) m=1;
    if(m<0) m=0;

    rgb[0]=(rgb[0]*(1−m))+(rgb2[0]*m);
    rgb[1]=(rgb[1]*(1−m))+(rgb2[1]*m);
    rgb[2]=(rgb[2]*(1−m))+(rgb2[2]*m);
}

void blending(gfloat *res, gfloat *A, gfloat *B, paint_brush *brush, gfloat r)
{
    guint i;
    gfloat blanc[3]={1,1,1};

    switch(brush−>blending_mode)
    {
    case NORMAL:
        for(i=0; i<3; ++i) res[i]=A[i]+(B[i]−A[i])*brush−>opacity*r;
        break;
    case ADD:
        for(i=0; i<3; ++i) res[i]=(B[i]*brush−>opacity*r)+A[i];
        break;
    case SUB:
        for(i=0; i<3; ++i) res[i]=A[i]−B[i]*brush−>opacity*r;
        break;
    }

    if(res[0]<0.0f) res[0]=0.0f; if(res[0]>1.0f) res[0]=1.0f;
    if(res[1]<0.0f) res[1]=0.0f; if(res[1]>1.0f) res[1]=1.0f;
    if(res[2]<0.0f) res[2]=0.0f; if(res[2]>1.0f) res[2]=1.0f;
}

gfloat CubicSpline(gfloat low, gfloat hight, gfloat pos)
{
    if(pos < low)
        return 0.0f;
    else
    {
        if(pos>=hight)
            return 1.0f;
    }
    pos = (pos − low)/(hight−low);
    return (3−2 * pos) * pos * pos;
}

/**
```

```
* applique un pinceau au coord (x,y) (+taille de pinceau)
*
**/
void apply_brush(paint_document *doc, gint x, gint y, gfloat *color)
{
   gfloat tmp[3];
   gfloat dest[3];

   guint size=paint_get_instance()->brush.size;

   gint startx;
   gint starty;
   gint endx;
   gint endy;
   gint i,j;

   /* on deborde ? */
   if(x<0 || x>doc->width)
      return;
   if(y<0 || y>doc->height)
      return;
   /* on clamp */
   startx=x-size;
   clamp(startx, 0, doc->width);
   endx=x+size;
   clamp(endx, 0, doc->width);
   starty=y-size;
   clamp(starty, 0, doc->height);
   endy=y+size;
   clamp(endy, 0, doc->height);

   /* on dessine */
   for(j = starty; j < endy; ++j)
   {
      for(i = startx; i < endx; ++i)
      {
         doc_get_color(doc, i,j, tmp);

         gfloat sx=x-i;
         gfloat sy=y-j;
         gfloat d=sqrtf(sx*sx + sy*sy);
         gfloat r=CubicSpline(0,size, d);

         blending(dest, tmp, color, &paint_get_instance()->brush, 1-r);
         doc_set_color(doc, i,j, dest);
      }
   }
}

/* PRESS */
gboolean paint_document_button_press(GtkWidget *w, GdkEventButton *e, gpointer data)
```

```
{
   gint x;
   gint y;
   GdkModifierType state;

   paint_document *doc=(paint_document*)data;

   if(e->button == 1 && doc->buffer != NULL)
   {
      gdk_window_get_pointer (w->window, &x, &y, &state);
      apply_brush(doc, x,y, paint_get_instance()->brush.color_a);
      gtk_widget_queue_draw(w);
   }
   if(e->button == 3 && doc->buffer != NULL)
   {
      gdk_window_get_pointer (w->window, &x, &y, &state);
      apply_brush(doc, x,y, paint_get_instance()->brush.color_b);
      gtk_widget_queue_draw(w);
   }
   return TRUE;
}

/* RELEASE */
gboolean paint_document_button_release(GtkWidget *w, GdkEventButton *e, gpointer data)
{
   /* pour l'instant vide */
   return TRUE;
}

/* MOTION */
gboolean paint_document_motion_notify(GtkWidget *w, GdkEventMotion *e, gpointer data)
{
   gint x;
   gint y;
   GdkModifierType state;

   paint_document *doc=(paint_document*)data;

   if(e->is_hint)
      gdk_window_get_pointer(w->window, &x, &y, &state);
   else
   {
      x=e->x;
      y=e->y;
      state=(GdkModifierType)e->state;
   }

   if((state & GDK_BUTTON1_MASK) && doc->buffer != NULL)
   {
      apply_brush(doc, x,y, paint_get_instance()->brush.color_a);
      gtk_widget_queue_draw(w);
```

```
    }
    else if((state & GDK_BUTTON3_MASK) && doc->buffer != NULL)
    {
       apply_brush(doc, x,y, paint_get_instance()->brush.color_b);
       gtk_widget_queue_draw(w);
    }
    return TRUE;
}
```

main.cc

```
#include <gtk/gtk.h>
#include "paint.h"

int main(int argc,char **argv)
{
    gtk_init(&argc,&argv);
    /** création */
    paint_create();

    gtk_main();

    /** destruction */
    paint_delete();

    return (0);
}
```