Les définitions `I2C_RELAY` et `I2C_INPUT` sont les adresses de nos 2 `Ioexpander`, adresse qui peut être paramétrée de façon matérielle (un périphérique possède généralement une adresse de base et la possibilité de modifier les bits de point faible à l'aide de 3 broches dédiées).

Voici donc le minimum qui nous permettra de compiler et d'exécuter le programme sur la cible. Passons à présent à la création de nos objets…

Initialisation graphique

Une image à l'écran (au sens Nano-X du terme) est définie autour de 3 valeurs essentielles :

- Position X,
- Position Y,
- Identifiant de l'image.

De plus, notre application très simple sera composée d'un fond d'écran et de 2 types d'images pour les Leds (images ON et OFF) et les relais (Listing 4).

Nous allons modifier la fonction `main()` afin d'utiliser la librairie graphique, créer une fenêtre de base et initialiser les objets de notre application (Listing 5). Comme vous avez pu le remarquer, l'initialisation de l'interface Nano-X est très simple : il suffit de créer une fenêtre (`MainWindow`) pour laquelle on active certains événements, plus on lui associe un contexte graphique (gc). Ces 2 variables seront nécessaires à presque toutes les fonctions de la librairie Nano-X, car il importe de spécifier à quelle fenêtre correspond une action (Nano-X est un serveur multi-fenêtres) et à quel contexte graphique (plusieurs GC peuvent être affectés à une même fenêtre).

Comme nous ne cherchons à réaliser une interface multi fenêtre, et que nous utiliserons pas les possibilités graphiques pour fermer une fenêtre (le croix en haut à droite d'une fenêtre), nous pourrions nous passer de la sélection des évènements. Toutefois nous avons préféré la laisser ici pour des raisons de compatibilité et pour vous permettre éventuellement de passer à une gestion de plusieurs fenêtres.

Premier affichage

Tout est définitivement prêt pour enfin afficher notre application graphique (Figure 5), c'est-à-dire notre fond d'écran, nos 3 relais et nos 3 leds (Listing 6).

Ajoutons l'appel à la fonction `SetupScreen()` juste avant `SetupTouchscreen`. Pour afficher

Listing 8. Écriture sur le bus I2C

```
void I2CWrite(int addr, char data) {
   int ret;
   ret = ioctl(fd_I2C, I2C_SLAVE_FORCE, addr);
   if (ret < 0) {
      printf("ioctl I2C_SLAVE 0x%02x returns %d\n", addr, ret); exit(1);
   }
   ret = write(fd_I2C, &data, 1);
   if (ret == -1) {
      perror("Erreur ecriture...\n");
      return;
   }
}
```

Listing 9. Lecture du bus I2C

```
char I2CRead(int addr) {
   int ret;
   char data;
   ret = ioctl(fd_I2C, I2C_SLAVE_FORCE, addr);
   if (ret < 0) {
      printf("ioctl I2C_SLAVE 0x%02x returns %d\n", addr, ret); exit(1);
   }
   ret = read(fd_I2C, &data, 1);
   if (ret == -1) {
      perror("Erreur ecriture...\n");
      return -1;
   }
   return data;
}
```

Listing 10. Boucle principale

```
while(1) {
   ret = read(fd_TS, &ts, sizeof(struct ts_event));
   if (ret > 0) {
      X = (ts.y * 320);
      Y = (ts.x * 240);
      X = X / 255;
      Y = Y / 255;
      Y = 255 - Y;
      //printf(" read TS (%d) x=%d, y=%d\n", ret, ts.x, ts.y);
      if (Y > 40 && Y < 140) {
      if (X > 15 && X < 95)
         ToggleRelay(0);
      if (X > 115 && X < 195)
         ToggleRelay(1);
      if (X > 215 && X < 295)
         ToggleRelay(2);
      }
   usleep(200000);
   }

   /* We flush the TS buffer */
   do ret = read(fd_TS, &ts,
         sizeof(struct ts_event));
   while (ret > 0);
   usleep(100000);
}
```