POO & C++ : TP 5

EISE4 2014—2015

Pablo Rauzy

rauzy@enst·fr

pablo.rauzy.name/teaching.html#epu-cpp

22 octobre 2014

N'oubliez pas :

- Les TPs doivent être rendus par courriel à `rauzy@enst·fr` au plus tard le lendemain du jour où ils ont lieu avec `[EISE4]` suivi de vos noms dans le sujet du mail.
- Le code rendu doit être propre !
- Le TP doit être rendu dans une archive tar gzippée : `tar czvf NOMS.tgz NOMS` où `NOMS` est le nom du répertoire dans lequel il y a votre code (idéalement, les noms des gens du groupe) ; attention à ne pas rendre un répertoire avec des fichiers temporaires dus à la compilation (pensez à faire un `make clean` avant `tar`).
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettrez dans un fichier texte `reponses.txt`.
- Le code est de préférence en anglais, avec des commentaires en français ou anglais, en restant cohérent.
- Le code doit être proprement indenté.
- Respectez les conventions de nommage :
  - variables : `nom_explicite` (*e.g.*, `game_type`),
  - fonctions : `verbeAction` (*e.g.*, `launchNewGame`),
  - classes : `NomExplicite` (*e.g.*, `BoardGame`),
  - membres privés ou protégés : comme variable ou fonction + `_` (*e.g.* `max_number_of_player_`, ou `updateState_`),
  - accesseurs d'affectation : `set_` + nom de l'attribut sans le `_` final (*e.g.*, `set_max_number_of_player`),
  - accesseurs de consultation : nom de l'attribut sans le `_` final (*e.g.*, `max_number_of_player`),
  - constantes : `NOM_EXPLICITE` (*e.g.*, `VERSION_NUMBER`).
- Pensez à utiliser les outils de développement comme le debugger `gdb` ou le profiler `valgrind`.
- **N'hésitez jamais à chercher de la documentation par vous-mêmes sur le net !**
  Par exemple la STL est documentée ici : `http://en.cppreference.com/w/cpp`,
  et la SFML l'est ici : `http://sfml-dev.org/documentation/1.6/`.

Dans ce TP :

- Utilisation de la SFML pour créer un logiciel de dessin minimaliste.

Exercice 0.

Récupération des fichiers nécessaires.

1. Pensez à organiser correctement votre espace de travail, par exemple tout ce qui se passe dans ce TP devrait être dans `~/cpp/tp5/`.

2. Récupérez les fichiers nécessaires au TP sur la page du cours.

3. Une fois que vous l'avez extrait de l'archive (`tar xzf tp5.tgz`), renommez le répertoire `tp5` en les noms de votre groupe (par exemple en `Dupont-Dupond`). Si vous ne le faites pas tout de suite, pensez à le faire avant de rendre votre TP.

Exercice 1.

Dans l'état où il vous est fourni, le code compile et une fenêtre s'affiche, mais on ne peut rien faire avec, même pas la fermer. Pour cause aucun évènement utilisateur n'est géré.

Le but de cet exercice va être de mettre en place la gestion des évènements.

Un `Makefile` vous est fourni, profitez en pour compiler votre code régulièrement (au minimum à la fin de chaque question) pour être sûr que tout va bien. N'hésitez pas non plus à lancer l'exécutable obtenu (`./minipaint`) pour jouer avec et le voir avancer.

1. Explorez le code existant et essayez de comprendre un peu son organisation. On a 5 modules.

- Le module `main` ne fait rien d'intéressant, il se contente de créer une instance de `MiniPaint` et de la lancer.
- Le module `minipaint` contient le code principale du logiciel. La boucle principale se trouve dans la méthode `MiniPaint::run`. Parmi ses membres se trouve la fenêtre principale du programme (`sf::RenderWindow`), mais aussi des composants de l'interface : la palette de couleur, les outils, la barre de statut, et une liste de stockage des formes géométriques (`sf::Shape`) qui composeront le dessin.
- Le module `palette` qui servira à gérer la palette de couleur.
- Le module `tools` qui servira à gérer les outils.
- Le module `statusbar` qui sert à gérer la barre de statut.

Ce qui nous intéresse pour l'instant est la méthode `MiniPaint::run`. Celle-ci contient la boucle principale du programme mais pas encore la boucle de gestion des évènements.

→ Ajoutez la boucle de gestion des évènements et faites en sorte qu'on puisse fermer la fenêtre (appeler la méthode `sf::RenderWindow::Close` lors de l'évènement `sf::Event::Closed`). Vous allez pour cela avoir besoin de déclarer une variable de type `sf::Event`.

Réponse :

```
sf::Event event;
while (win_->IsOpened()) {
  while (win_->GetEvent(event)) {
      switch (event.Type) {
      case sf::Event::Closed:
        win_->Close();
        break;
      default: break;
    }
  }
  //...
}
```

2. On veut maintenant dessiner lorsque l'utilisateur clique avec la souris.

Pour cela, on va appeler la méthode `setPosition` de la classe `Tools` (sur l'instance membre de la classe `MiniPaint`) lors de l'évènement `sf::Event::MouseButtonPressed` correspondant à l'appui sur un bouton de la souris. L'appel à `setPosition` signifie que l'on est à présent en train de dessiner un nouvel élément du dessin. On va passer à cette méthode les coordonnées du clic qui seront celles du nouvel élément. On peut récupérer ces coordonnées dans les champs `.MouseButton.X` et `.MouseButton.Y` de l'évènement.

Il faut aussi ajouter le nouvel élément dans la liste des formes du dessin lorsque l'utilisateur relâche la souris (évènement `sf::Event::MouseButtonReleased`) si celui-ci était en train de dessiner (*i.e.*, si `tools_->isDrawing()` est vrai).

Pour récupérer le nouvel élément, on appelle la méthode `makeShape` de la classe `Tools` avec comme premier argument la couleur voulue (c'est à dire `palette_->getColor()`) et comme second argument `true` pour lui signifier que cet élément du dessin est fini.

→ Ajoutez les deux nouveaux `case` au `switch` de la boucle d'évènements.

Réponse :

```
case sf::Event::MouseButtonPressed:
  tools_->setPosition(event.MouseButton.X, event.MouseButton.Y);
  break;
case sf::Event::MouseButtonReleased:
  if (tools_->isDrawing()) {
    shapes.push_back(tools_->makeShape(palette_->getColor(), true));
  }
  break;
```

3. Maintenant on veut implémenter le changement des outils. L'idée est que quand l'utilisateur appui sur la touche majuscule et une autre touche en même temps, cela correspond à un changement d'outil : majuscule + L pour les lignes, majuscule + C pour les cercles, majuscule + R pour les rectangles, et majuscule + P pour la palette de couleur.

Comme vous pouvez le voir, il y a déjà dans la classe `Tools` une méthode permettant de faire ce changement qui reçoit en argument une touche du clavier telle qu'encodée par SFML (du type `sf::Key::Code`).

Il nous faut donc appeler cette méthode quand une touche est enfoncée (évènement `sf::Event::KeyPressed`) et que la touche majuscule l'est aussi. Pensez à utiliser la classe `sf::Input` comme on l'a vu dans le cours pour cela.

→ Ajoutez la nouvelle clause `case` dans le `switch` de la boucle d'évènements.

Réponse :

```
case sf::Event::KeyPressed:
  if (input.IsKeyDown(sf::Key::LShift)) {
    tools_->switchTool(event.Key.Code);
  }
  break;
```

Exercice 2.

Comme vous avez pu le constater à la fin de l'exercice précédent, les outils ne sont pas vraiment implémenté et on peut seulement faire des points noirs pour l'instant.

Le but de cet exercice est de faire marcher les trois outils de dessin : ligne, cercle, rectangle.

1. Référez-vous à la documentation ou au tutoriel de la SFML pour trouver comment faire des lignes, cercles, et rectangles (il y a des fonctions toutes prêtes).

Remarquez que la classe `Tools` a un pointeur sur la fenêtre principale comme membre, dont vous pouvez vous servir pour récupérer une instance de `sf::Input`, par exemple.

→ Dans le fichier `tools.cpp`, complétez la méthode `makeShape`.

Réponse :

```
case Line:
  s = sf::Shape::Line(x_, y_, input.GetMouseX(), input.GetMouseY(), 1.0, c);
  break;
case Circle:
  s = sf::Shape::Circle(x_, y_, sqrt(pow(input.GetMouseX() - x_, 2) +
                                     pow(input.GetMouseY() - y_, 2)), c);
  break;
case Rectangle:
  s = sf::Shape::Rectangle(x_, y_, input.GetMouseX(), input.GetMouseY(), c);
  break;
```

2. Votre logiciel de dessin sait maintenant faire des lignes, des cercles et des rectangles ! Il suffit pour cela de changer d'outil avec les raccourcis clavier définis précédemment et d'utiliser la souris pour tracer l'élément. Par contre, les éléments ne sont affichés qu'une fois qu'on a relâché la souris, ce qui n'est pas très pratique. On voudrait que l'élément en cours de création soit affiché aussi.

→ Dans la méthode `MiniPaint::run`, juste avant l'instruction `win_->Display()`, faites en sorte d'afficher, si on est en train de dessiner, l'élément en cours de création, que l'on peut récupérer avec un appel à la méthode `makeShape` de `Tools` comme dans la question 2 de l'exercice 1, mais sans le second argument (ou alors en lui donnant la valeur `false`) pour indiquer que le dessin de l'élément en cours de création n'est pas terminé.

Réponse :

```
if (tools_->isDrawing()) {
  win_->Draw(tools_->makeShape(palette_->getColor()));
}
```

3. Essayez de redimensionner la fenêtre de MiniPaint après avoir dessiné un peu dedans. Observez ce qu'il se passe. Ce n'est pas satisfaisant : lorsqu'on redimensionne la fenêtre on ne veut pas que le dessin soit déformé, on veut simplement changer la taille de la surface de dessin.

→ Adaptez la vue de la fenêtre en appelant la méthode `sf::RenderWindow::SetView` lors de l'évènement `sf::Event::Resized`.

Réponse :

```
case sf::Event::Resized:
  win_->SetView(sf::View(sf::FloatRect(0, 0, event.Size.Width, event.Size.Height)));
  break;
```

Exercice 3.

Dans cet exercice on va implémenter le choix de la couleur de dessin.

1. La stratégie que l'on va utiliser pour le choix des couleurs est d'afficher une image contenant plein de couleurs et de permettre à l'utilisateur de choisir une couleur en cliquant dessus.

Pour cela ils nous faut donc charger en mémoire une telle image. Il y en a une, `color-picker.png` qui vous est fournie, mais vous pouvez en choisir une autre si vous préférez (ne perdez pas de temps pour ça cela dit).

→ Dans le constructeur de `Palette`, faites pointer le membre `colorpicker_` vers une instance de `sf::Image`. Dessus, utilisez la méthode `LoadFromFile` pour charger l'image en mémoire puis la méthode `Bind` pour la préparer pour l'affichage.

Réponse :

```
colorpicker_ = new sf::Image;
colorpicker_->LoadFromFile("color-picker.png");
colorpicker_->Bind();
```

2. → Toujours dans le même constructeur, faites pointer le membre `palette_` vers une instance de `sf::Sprite` dont l'image sera celle pointée par `colorpicker_`.

Réponse :

```
palette_ = new sf::Sprite(*colorpicker_);
```

3. → N'oubliez pas de faire les appels à `delete` nécessaires dans le destructeur de `Palette`.

Réponse :

```
delete palette_;
delete colorpicker_;
```

4. On veut maintenant implémenter la méthode `Palette::display`. Le but est d'afficher la palette de couleurs au milieu de la fenêtre [1], avec tout autour un voile sombre, et en dessous un rectangle avec la couleur actuellement survolée et un autre avec la couleur actuellement sélectionnée.

Pour positionner la palette, on utilise la méthode `sf::Sprite::SetPosition`. La position adéquat se calcule à partir de la taille de la palette et des dimensions de la fenêtre (récupérable avec `sf::RenderWindow::GetWidth`, `sf::RenderWindow::GetHeight`, et la classe `Palette` contient un pointeur vers la fenêtre principale).

Pour le voile sombre, il suffit de faire un rectangle noir transparent qui recouvre toute la fenêtre. La transparence peut s'obtenir avec le 4ème paramètre (alpha) du constructeur `sf::Color`.

Les coordonnées des rectangles d'information se calculent à partir de la taille de la palette, des dimensions de la fenêtre et bien sûr de leur taille.

La couleur du premier rectangle est la valeur de l'attribut `color_` de la classe `Palette`, la couleur du second est le résultat de l'appel à la méthode `getColor` prenant la position de la souris en paramètre (pensez à `sf::Input`).

→ Implémentez la méthode `Palette::display`.

Réponse :

```
int w = win_->GetWidth(), h = win_->GetHeight() - 20;
palette_->SetPosition(w / 2 - 128, h / 2 - 138);
win_->Draw(sf::Shape::Rectangle(0, 0, w, h, sf::Color(0, 0, 0, 200)));
win_->Draw(*palette_);
win_->Draw(sf::Shape::Rectangle(w / 2 - 128, h / 2 + 118, w / 2, h / 2 + 138, color_));
win_->Draw(sf::Shape::Rectangle(w / 2, h / 2 + 118, w / 2 + 128, h / 2 + 138,
                                getColor(win_->GetInput().GetMouseX(),
                                         win_->GetInput().GetMouseY())));
```

5. Il faut maintenant faire appel à la méthode que l'on vient d'implémenter. Dans `MiniPaint::run`, juste après l'affichage conditionnel de l'élément en cours de dessin, il faut appeler la méthode `display` de l'attribut `palette_` si l'outil actif est la palette. On peut récupérer l'outil actif en utilisant la méthode `getActiveTool` de `Tools`.

→ Ajoutez l'affichage de la palette lorsque celle-ci est l'outil actif.

1. Cette question et la 7 seront plus simple si vous choisissez de simplement afficher la palette dans le coin en haut à gauche de la fenêtre, c'est à dire aux coordonnées $(0, 0)$.

Réponse :

```
if (tools_->isDrawing()) {                                    // déjà là
  win_->Draw(tools_->makeShape(palette_->getColor()));        // déjà là
}                                                             // déjà là
else if (tools_->getActiveTool() == Tools::Palette) {
  palette_->display();
}
```

6. La palette ne marche pas encore, mais elle peut maintenant s'afficher. Avant de la faire vraiment fonctionner, on veut pouvoir la faire disparaître pour continuer à dessiner.

→ Dans le cas d'un appui sur une touche, vérifiez si cette touche n'est pas Échap (`sf::Key::Escape`) et dans ce cas, si l'outil actif est la palette, appelez la méthode `switchBack` de `Tools` pour revenir à l'outil précédent.

Réponse :

```
case sf::Event::KeyPressed:
  if (input.IsKeyDown(sf::Key::LShift)) {                     // déjà là
    tools_->switchTool(event.Key.Code);                       // déjà là
  }                                                           // déjà là
  else switch (event.Key.Code) {
    case sf::Key::Escape:
      if (tools_->getActiveTool() == Tools::Palette) {
        tools_->switchBack();
      }
      break;
    default: break;
  }
  break;                                                      // déjà là
```

7. Maintenant, on souhaite faire fonctionner la palette, c'est à dire commencer par implémenter la méthode `getColor(int,int)` de `Palette` qui pour l'instant renvoie simplement la couleur noir.

Vous pouvez récupérer la position d'un `sf::Sprite` avec sa méthode `GetPosition`. Vous pouvez récupérer la couleur d'un de ses pixels avec la méthode `GetPixel`.

Si les coordonnées passées en argument ne sont pas sur la palette, alors retourner la couleur courante.

→ Implémentez la méthode `getColor(int, int)` de la classe `Palette`.

Réponse :

```
sf::Vector2f pos = palette_->GetPosition();
sf::Color c = color_;
if (x >= pos.x && x < pos.x + 256 && y >= pos.y && y < pos.y + 256) {
  c = palette_->GetPixel(x - pos.x, y - pos.y);
}
return c;
```

8. La palette est maintenant capable d'afficher la couleur en cours de survol, mais pas de la sélectionner. Il faut pour cela mettre à jour la gestion de l'évènement `sf::Event::MouseButtonPressed` dans `MiniPaint::run`.

Si l'outil actif est la palette, alors appeler la méthode `updateColor` de `Palette` en lui donnant en paramètre les coordonnées de la souris associées à l'évènement, sinon faire le traitement déjà en place (pour commencer à dessiner un élément).

→ Gérez le changement de couleur.

Réponse :

```
if (tools_->getActiveTool() == Tools::Palette) {
  palette_->updateColor(event.MouseButton.X, event.MouseButton.Y);
}
else {
  tools_->setPosition(event.MouseButton.X, event.MouseButton.Y);       // déjà là
}
```

Exercice 4.

Cet exercice est un bonus. Il est très peu guidé.

1. → Affichez la barre de statut.

Réponse :

```
status_bar_->display(); // juste avant win_->Display();
```

2. → Implémentez la sauvegarde de l'image lors de l'appui sur ctrl+s (toujours dans le même fichier `saved.png`).

Réponse :

```
case sf::Event::KeyPressed:
  if (input.IsKeyDown(sf::Key::LShift)) {
    tools_->switchTool(event.Key.Code);
  }
  else if (input.IsKeyDown(sf::Key::LControl)) {          // nouveau
    switch (event.Key.Code) {                             // nouveau
    case sf::Key::S:                                      // nouveau
      win_->Capture().SaveToFile("saved.png");            // nouveau
      break;                                              // nouveau
    default: break;                                       // nouveau
    }                                                     // nouveau
  }                                                       // nouveau
  else switch (event.Key.Code) {
    case sf::Key::Escape:
      if (tools_->getActiveTool() == Tools::Palette) {
        tools_->switchBack();
      }
      break;
    default: break;
  }
  break;
```

3. → Implémentez les opérations annuler / refaire sur ctrl+z et ctrl+y.

Réponse :

```
case sf::Event::KeyPressed:
  if (input.IsKeyDown(sf::Key::LShift)) {
    tools_->switchTool(event.Key.Code);
  }
  else if (input.IsKeyDown(sf::Key::LControl)) {
    switch (event.Key.Code) {
    case sf::Key::Z:                                    // nouveau
      if (!shapes.empty()) {                            // nouveau
        canceled_shapes.push_back(shapes.back());       // nouveau
        shapes.pop_back();                              // nouveau
      }                                                 // nouveau
      break;                                            // nouveau
    case sf::Key::Y:                                    // nouveau
      if (!canceled_shapes.empty()) {                   // nouveau
        shapes.push_back(canceled_shapes.back());       // nouveau
        canceled_shapes.pop_back();                     // nouveau
      }                                                 // nouveau
      break;                                            // nouveau
    case sf::Key::S:
      win_->Capture().SaveToFile("saved.png");
      break;
    default: break;
    }
  }
  else switch (event.Key.Code) {
    //...
  }
  break;
```

4. → Ajoutez la possibilité de bouger le dernier élément du dessin avec les touches fléchées.

Réponse :

```
case sf::Event::KeyPressed:
  //...
  else switch (event.Key.Code) {
    case sf::Key::Escape:
      if (tools_->getActiveTool() == Tools::Palette) {
        tools_->switchBack();
      }
      break;
    case sf::Key::Up:                 // nouveau
      shapes.back().Move(0, -1);      // nouveau
      break;                          // nouveau
    case sf::Key::Down:               // nouveau
      shapes.back().Move(0, 1);       // nouveau
      break;                          // nouveau
    case sf::Key::Left:               // nouveau
      shapes.back().Move(-1, 0);      // nouveau
      break;                          // nouveau
    case sf::Key::Right:              // nouveau
      shapes.back().Move(1, 0);       // nouveau
      break;                          // nouveau
    default: break;
  }
  break;
```