Projet : étape 6 (semaines 8 et 9 du semestre)

Buts

Nous allons cette semaine donner un aspect un peu plus visuel à notre projet. Concrètement, nous allons fournir un affichage graphique 3D de nos objets.

Préliminaires :

Je vous rappelle qu'il est important de savoir se répartir le travail au sein du groupe.

Comme cette semaine nous abordons le graphisme, il me semble opportun qu'un des deux membres du binôme se charge spécifiquement de cet aspect (pensez à vous partager le travail) et ait lu et pratiqué le **tutoriel sur le graphisme** présenté il y a presque deux mois.

Pensez également à tenir à jour votre fichier `JOURNAL` ainsi qu'à répondre périodiquement aux questions. Cela fait partie intégrante du projet.

---

[3] Exercice P10 : combiner la simulation et l'affichage 3D

P10.1 - Introduction (et mise en garde)

Le but de cette partie est donc d'ajouter des fonctionnalités graphiques à notre projet au moyen des bibliothèques OpenGL et Qt (ou autre, voir à ce sujet **l'introduction donnée dans l'exercice P2**).
Nous abordons donc ici un point valorisant, mais **difficile**, prenant et **chronophage** de notre projet. À la fin de cet exercice P10, vous devriez en effet *voir* vos premiers points matériels s'agiter en 3D !

> **Remarque importante :**
>
> Comme dit, mettre en place le graphisme est *chronophage*. Avant de commencer cette partie, regardez bien **le barème** dans sa globalité, et **décidez** si cette partie (qui ne représente que quelques pour-cent du barème total) fait partie de vos objectifs. On peut très bien continuer la suite du projet *sans* partie graphique (c'est pour cela que le niveau de cet exercice n'a pas d'étoile).
>
> Autrement dit : sauf si c'est vraiment une partie qui vous intéresse/que vous voulez avoir, je déconseille à des groupes visant moins de 5.5 sur le projet et déjà en retard à ce stade de se lancer dans ce sujet.

P10.2 - Tutoriel : graphisme et programmation orientée objet

NOTE : cette partie ne devrait concerner que celui/celle en charge de l'aspect graphique.

Au niveau de la conception, l'idée est de pouvoir garder l'ancienne version non-graphique en parallèle à la version graphique. On suit pour cela la démarche expliquée dans le deuxième exemple du **tutoriel sur le graphisme** ainsi que dans **l'exercice P8**.

Dans cet exercice ci, je vous propose de repartir **du code de l'exemple 5 du tutoriel**, et de le faire évoluer vers une version qui devrait ressembler à celle que vous utiliserez pour votre projet. C'est donc ici une étape pédagogique intermédiaire. Celles et ceux qui ont déjà bien compris le tutoriel et/ou connaissent déjà bien le graphisme peuvent sans soucis sauter cette partie du tutoriel et passer directement à **la section suivante**.

Créez un répertoire `exerciceP10tuto` sur la base de `ex_05` du **cinquième exemple du tutoriel sur le graphisme [fichier zip]**.
Concrètement :

- si nécessaire, relisez le tutoriel ; comprenez bien son esprit et le rôle de chaque classe ;

- dézipez le fichier ci-dessus, renommez le répertoire `ex_05` en `exerciceP10`, puis renommez le fichier `ex_05.pro` en `exerciceP10.pro`.

Créez dans la partie générale (c.-à-d. dans le répertoire `general`) une classe `Machin` qui hérite de la classe abstraite `Dessinable` et qui contient simplement en plus une méthode publique virtuelle pure `void evolue(double dt);`.
Cette classe étant une pure abstraction, il **n'**est **pas** nécessaire d'y définir la méthode `dessine_sur()`.
À noter aussi qu'un fichier `.h` suffit (il n'y a pas besoin de `.cc` ; il n'y aurait rien à y mettre...).

Créez ensuite une classe `QuiTourne` qui hérite de la classe abstraite `Machin` et qui contiendra un angle. Il suffit pour cela de *copier* `contenu.h` en `quitourne.h` et de *renommer* `contenu.cc` en `quitourne.cc`, puis éditer ces deux fichiers pour renommer les classes (mère et fille, y compris les `include`).
Changez aussi le nom `contenu` par `quitourne` dans `general.pro`.
Profitez-en aussi pour changer un peu le constructeur et passer l'angle comme paramètre (en gardant la valeur par défaut 0.0).
Enfin, cette classe étant aussi abstraite, on peut y supprimer la redéfinition de la méthode `dessine_sur()`.

Créez une classe `Moucheron` qui hérite de `QuiTourne`. Ici comme on fait un exemple très simple il n'y a rien d'autre à faire qu'un fichier `.h` (pas besoin de `.cc`) avec simplement la définition usuelle de la méthode `dessine_sur()`.

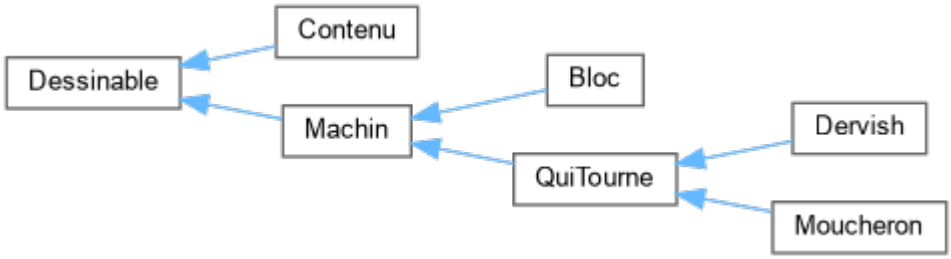
Faites de même avec une classe `Dervish`.

Note : encore une fois, on fait ici un exemple très simple qui ne différenciera ces deux classes que dans leur affichage. Dans un vrai gros projet, elles seraient bien sûr plus différentes et nécessiteraient certainement des `.cc` ; mais ce n'est pas notre propos ici.

Créez ensuite une classe `Bloc` qui hérite de la classe abstraite `Machin` et ayant une redéfinition de la méthode `void evolue(double dt);` qui ne fait simplement rien (cet objet ne bouge pas).
Elle a par contre la définition usuelle de la méthode `dessine_sur()`.
Et là non plus pas besoin de `.cc`, le `.h` devrait suffire.

On va maintenant adapter la classe `Contenu` pour qu'elle soit une collection hétérogène de `Machin`.
Là aussi c'est assez simple pour ne le faire que dans le `.h` (pas besoin de `.cc`.
Transformez donc `Contenu` en une collection hétérogène de `Machin`, de la façon qui vous convient le mieux.
Supprimez, bien sûr, l'attribut d'angle.
Cette classe aura simplement trois méthodes :
une `dessine_sur()` et une méthode `evolue()`, lesquelles sont contenteront simplement d'appeler les méthodes correspondant de chacun des éléments de la collection ;
et une méthode `ajoute()` pour ajouter un `Machin` à la collection (choisissez le moyen qui vous convient le mieux).

On arrive donc au final au graphe d'héritage suivant:



où `Contenu` est une collection hétérogène de `Machin`.

Il ne reste plus pour finir qu'à adapter nos `SupportADessin` (les classes `TextViewer` et `VueOpenGL`), ainsi que les « `main()` » :

- supprimez

  ```
  void dessine(Contenu const& a_dessiner) override;
  ```

  et ajoutez

  ```
  void dessine(Bloc      const& a_dessiner) override;
  void dessine(Moucheron const& a_dessiner) override;
  void dessine(Dervish   const& a_dessiner) override;
  ```

  aux classes `SupportADessin` (là en virtuelles pure et sans `override`, bien sûr !), `TextViewer` et `VueOpenGL` ;
- définissez le « dessin » de ces trois classes en mode texte ; p.ex. :

```
void TextViewer::dessine(Moucheron const& a_dessiner)
{
  flot << "Moucheron : " << a_dessiner.infos() << std::endl;
}

void TextViewer::dessine(Dervish const& a_dessiner)
{
  flot << "Devish : " << a_dessiner.infos() << std::endl;
}

void TextViewer::dessine(Bloc const& a_dessiner __attribute__((unused)))
{
  flot << "Bloc : immobile" << std::endl;
}
```

- Dans `vue_opengl.cc`, séparez en trois la méthode `dessine()` :
  - gardez le dessin des deux premiers cubes dans le `dessine(Bloc const&);`
  - mettez le dessin du troisième cube dans le `dessine(Dervish const&);`
    et changez le `45.0` en `45.0 + 3.0 * a_dessiner.infos()`
  - et mettez le dessin du quatrième cube dans le `dessine(Moucheron const&);`
- dans `main_text.cc`, ajoutez un `Bloc`, un `Moucheron` et un `Dervish` au `Contenu` ;
- et dans `glwidget.cc`, déplacez le constructeur depuis le `.h` et ajoutez également un `Bloc`, un `Moucheron` et un `Dervish` à `c` (dans le corps du constructeur).

Voilà ! Vous devriez maintenant avoir un programme qui fait presque la même chose que l'exemple 5 du tutoriel graphique (sauf que le petit cube de droite tourne aussi), mais qui a une structure plus simple à adapter à votre projet.

Amélioration : vous pourriez aussi en profiter pour passer la méthode `dessine_sur()` en `const`.

P10.3 - Combiner la simulation et l'affichage 3D

Nous arrivons maintenant à un point délicat de notre projet : intégrer la simulation dans l'interface graphique ! A la fin de cet exercice, vous devriez en effet voir vos premiers points matériels bouger en 3D !

Pour cela, commencez par recopier votre répertoire `exerciceP10a` précédent en un `exerciceP10` (sans 'a').

Supprimez ensuite les classes jouets qui ne sont plus nécessaire et adaptez le reste de votre projet en dessinant pour chaque point matériel spécifique une forme *simple* (cube, cône, sphère, tore, ...) centrée sur les coordonnées du point matériel.

Réfléchissez à ce que doivent être les classes `Contenu/Machin` : faut-il les garder ? la compléter/modifier ? voire carrément les remplacer ? par lesquelles de vos classes à vous ?
En bref, faites ici la connexion conceptuelle entre le tutoriel ci-dessus et votre projet développé jusqu'ici.

- Déplacez/Recopiez le contenu de votre projet dans `exerciceP10/general` et mettez à jour le fichier `.pro` ;

- si ce n'est pas déjà fait comme cela, commencez par la version texte (`TextViewer`), c.-à-d. par reproduire **l'exercice P9** dans ce nouveau cadre ;

mettez pour cela à jour les classes `SupportADessin` et `TextViewer` (ainsi que `main_text.cc` et `text.pro`) ;

testez la version en mode texte ;

passez ensuite à la version graphique : ajoutez les dessins nécessaires dans la classe `VueOpenGL`.

Si ce n'est pas déjà fait (cf exercice P10 ci-dessus), intégrez toutes les variables nécessaires comme attributs de la classe `GLWidget` et associez la méthode `Systeme::evolue()` au «timer» (`GLWidget::timerEvent()`).

Attention aussi à bien ajuster les paramètres d'échelle du graphisme de sorte à voir effectivement quelque chose ; c.-à-d. de sorte que les objets dessinés apparaissent dans la fenêtre.

Dans un premier temps, ne liez pas le temps réel (temps physique de l'interface) au temps de simulation, mais utilisez simplement un pas de temps fixe (le mieux serait que ce soit un attribut de la classe `Systeme`).

Dans un second temps et si vous le souhaitez (optionnel), vous pouvez faire que ce «pas de temps» soit en fait un rapport entre l'écoulement du temps physique (de l'interface) et le temps de la simulation.
Choisissez cependant une unité de temps qui soit raisonnable de sorte à rendre la simulation à la fois réaliste et observable (c.-à-d. temps ralentit).

Si tout s'est bien passé, vous devriez voir vos premiers points matériels en mouvement. (Attention, il faudra peut-être

ajuster les paramètres d'échelle du graphisme de sorte à voir effectivement quelque chose ; c.-à-d. de sorte que les objets simulés soient dans la fenêtre).

[Question P10.1] Si vous souhaitez voir un champ ou encore une contrainte, c.-à-d. la faire dessiner, comment devez vous modifier (ou pas) votre conception/votre code ?
Répondez à cette question dans votre fichier `REPONSES`.

Sauvegardez bien cette étape du projet (tout le répertoire `exerciceP10`). Elle devra faire partie du rendu final.

Dernière mise à jour le 7 février 2025
Last modified: Fri Feb 7, 2025