Nicolas Ward
Frédéric Myotte

Fichier Réponse

Projet de Programmation

Session 2015

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Question P1.1

Les vecteurs que nous utiliserons tout au long du projet ont des dimensions variables. C'est pourquoi nous avons fait le choix de les représenter sous forme de tableaux dynamiques de nombres réels. Le seul attribut de notre classe Vecteur est donc un tableau dynamique V. L'attribut est privé. Il est muni des méthodes publiques getV - qui retourne le tableau V - et setV, qui permet de l'initialiser.

Toutes les méthodes demandées dans l'énoncé (compare, addition, multiplication scalaire, etc.) sont publiques. Autrement, elles ne seraient pas utilisables. Nous avons aussi ajouté une méthode "diminue", qui enlève une dimension à un vecteur. Elle nous sert essentiellement dans le fichier test.

Question P1.2

Pour toutes les opérations, dans le cas où les deux vecteurs n'ont pas la même dimension, nous avons choisi par convention que c'est la plus petite des deux qui détermine l'opération (par projection).
Initialement, nous avions considéré les vecteurs comme des tableaux dynamiques et nous voulions combler les cases du vecteur ayant la plus petite dimension avec des zéros, de manière à ce qu'il ait la même dimension que le second vecteur. Mais nous sommes tombés face à un problème. Malgré le fait que mathématiquement cela marche et ne change rien au vecteur, le fait de rajouter des zéros modifiait l'opérande, ce qui était contraire aux consignes de départ qui le pénalisaient.
Ainsi nous avons procédé autrement. Tout d'abord nous regardons si les dimensions des deux vecteurs sont différentes. Si c'est le cas, une nouvelle séparation des possibilités s'impose, que nous décrirons plus bas. Le cas échéant (les deux vecteurs ont la même dimension), la fonction s'effectuera normalement. Revenons au cas où les dimensions sont différentes. On regarde laquelle est la plus petite. Nous avons finalement ignoré les « cases en trop » du vecteur ayant la dimension la plus grande. Par exemple, si nous avons

deux vecteurs A et B de dimensions respectives 2 et 3, la dernière case du tableau B n'est pas prise en compte dans le calcul. Cette ignorance des éléments correspond mathématiquement à une projection, comme souhaité, et ne modifie pas les tableaux dynamiques représentant les vecteurs.

Question P4.1

Nous avons ajouté un constructeur de copie à notre programme. En effet, dans le cas où nous aurions besoin de créer un nouveau vecteur ne différant que d'une seule dimension par rapport à celui d'origine (à n dimensions), au lieu de copier manuellement toutes les (n-1) autres dimensions dans le nouveau vecteur, il nous suffira d'utiliser le constructeur de copie. Aussi dans le cas de l'utilisation d'un vecteur, on peut en faire autant de copies que nécessaire que l'on pourra modifier à notre guise sans changer celui-ci.

Question P4.2

a) Ajouter un constructeur qui utiliserait des coordonnées sphériques impliquerait une modification des attributs de la classe. En effet, il faudrait en ajouter trois, à savoir deux « double » pour les angles et un « unsigned double » pour la longueur (positive).

b) La difficulté majeure de ce choix de repère en C++ serait sans doute de coder les fonctions, qui reposent sur un repère cartésien. Ainsi toutes les formules mathématiques seraient à reprendre, il faudrait même sûrement réécrire de nouvelles fonctions propres à ce repère.

Question P4.3

Pour la méthode compare, nous avons introduit un opérateur interne (==) qui renvoie un booléen. Pour la méthode affiche, nous avons défini un opérateur externe d'affichage (<<). Dans les deux cas, nous avons fait appel aux méthodes correspondantes, respectivement "compare" et "affiche".

Question P5.1

Nous avons implémenté la masse comme attribut car nous n'avons pas besoin de la calculer: c'est une propriété fixe commune à toutes les boules.
Pour la masse volumique, nous avons défini une méthode, calculée grâce à une méthode volume() (privée). Nous avons choisi de la définir en tant que méthode car elle se calcule facilement à partir de nos attributs.

Question P5.2

Nous avons fait la même implémentation que dans l'énoncé: la position et l'angle sont dans le vecteur état, tandis que la vitesse et la vitesse angulaire sont dans le vecteur dérivée_temp.
Nous avons aussi défini des accesseurs et des modificateurs pour état et dérivée_temp, ainsi que pour la position, la vitesse et la vitesse angulaire. Nous n'en avons pas besoin pour l'angle.
Enfin, nous avons rajouté une méthode "modifie" dans Vecteur qui prend en paramètre la dimension que l'on souhaite modifier, ainsi que la valeur (double) qui remplacera l'ancienne dans cette dimension.

Question P6.1

"Integrateur" est une classe abstraite, avec une unique méthode "integre", virtuelle pure. Tous les intégrateurs que nous implémenterons par la suite (Euler-Crommer, Newmark, Runge-Kutta) hériteront de la classe intégrateur et posséderont leur propre méthode "intègre".

Question P6.2

Comme spécifié précédemment, IntegrateurEuler hérite de Integrateur.

Question P7.1

Boule est un Objet. Elle hérite donc des méthodes "distance", "collision" et "t_collision" en les définissant.

Question P8.1

La méthode "dessine" est virtuelle pure car elle est appelée de manière polymorphique. En effet, chaque Dessinable a son propre affichage.

Question P8.2

Pour gérer les chocs dans un système, on introduit deux collections hétérogènes: une d'objets mobiles et une d'objets immobiles. Les objets mobiles sont les boules.

Question P8.3

Il faut faire attention à trois choses en particulier. En premier, il ne faut pas faire de copie d'une classe de pointeurs. La solution est de mettre "delete" au constructeur de copie associé à la classe Systeme. Ensuite, il faut bien mettre les "delete" correspondant aux allocations dynamiques. Et enfin, il ne faut pas mettre de "delete" aux allocations statiques.

Question P8.4

La classe Systeme comporte deux collections hétérogènes d'Objets (mobiles et immobiles), ainsi qu'un pointeur sur un Integrateur.
Au niveau des méthodes, nous en avons deux pour ajouter des objets à nos collections hétérogènes. Nous avons aussi une méthode qui dessine le système et une autre qui le fait évoluer. Enfin, nous avons un constructeur et un destructeur.

Question P12.1

Pour faire disparaître une boule, nous n'avons pas fait appel au destructeur car il est sujet à trop d'incertitudes. Nous avons préféré implémenter une méthode "byebye" qui prend une boule B en paramètre et qui l'efface (delete B).
Cette méthode sera appelée dans la méthode "collision" de la classe Trou (lorsqu'une boule entre en collision avec un trou, elle disparaît).

Question P13.1

Les deux nouveaux intégrateurs héritent de notre classe "Intégrateur". ils seront modifiés dynamiquement au travers des "#include" en début de code.