***REPONSES***

*Ce document a pour but d’expliquer les choix effectués tout au long de notre conception*

*Semaine 2*

**[Question P1.1]** *Comment représentez-vous ces vecteurs ? Comment sont-ils organisés : quels attributs ? quelles méthodes ? quels droits d'accès ?*

Les vecteurs sont représentés par une nouvelle classe Vecteur2D dont les attributs sont les coordonnées selon x et y du vecteur, tous deux privés afin de préserver l’encapsulation – ils sont connus du programmeur mais ne peuvent pas être modifiés directement par l’utilisateur.

Nous lui avons ensuite ajouté plusieurs méthodes publiques permettant de modifier la valeur des coordonnées (ce qui affaiblit l’encapsulation), d’afficher une instance de la classe en affichant ses attributs et de tester l’égalité entre deux vecteurs en comparant leurs coordonnées à une précision près. Ces méthodes sont toutes trois publiques car l’utilisateur a besoin de les appeler directement afin de les utiliser dans son programme principal.

*Semaine 4*

**[Question P4.1]** *Avez-vous ajouté un constructeur de copie ? Pourquoi (justifiez votre choix) ?*

Nous avons ajouté un constructeur de copie, d’une part afin de rendre notre modèle de Vecteur2D plus complet et plus simple à utiliser (il est plus intuitif de copier directement un vecteur dans un autre plutôt que de copier chaque attribut un à un), et d’autre part afin de préserver l’encapsulation – en effet, il est préférable que ce soit le programme qui manipule les attributs pour les copier un à un plutôt que l’utilisateur.

**[Question P4.2]** *Si l'on souhaitait ajouter un constructeur par coordonnées polaires (un angle et une longueur),*

***a]*** *que cela impliquerait-il au niveau des attributs de la classe ?*

Cela n’impliquerait aucun changement majeur au niveau de la structure des attributs puisqu’en programmation orientée objet l’implémentation est séparée de l’interface, mais il serait judicieux de changer leur nom afin de mieux les associer à leur signification physique.

***b]*** *quelle serait la difficulté majeure (voire l'impossibilité) de sa réalisation en C++ ? (C'est d'ailleurs pour cela qu'on ne vous demande pas de faire un tel constructeur !)*

Un constructeur en coordonnées polaires provoquerait une confusion au moment de la construction des instances puisqu’il prendrait exactement les mêmes arguments qu’un constructeur en coordonnées cartésiennes et il serait alors impossible de séparer les deux types d’initialisation – une telle conception nous forcerait à changer la signature du nouveau constructeur (en rajoutant un nouveau paramètre inutile par exemple), ce qui impliquerait une difficulté inutile pour la compréhension de l’utilisateur.

**[Question P4.3]** *Quels opérateurs avez-vous introduits ?*

Afin de remplacer les méthodes affiche() et compare() nous avons introduit deux opérateurs :

* Le premier est l’opérateur externe << permettant d’afficher un vecteur en affichant ses coordonnées ;
* Le second est l’opérateur interne == permettant de comparer l’égalité entre deux vecteurs en comparant leurs coordonnées deux à deux à 10-10 près.

Nous avons par la suite ajouté des opérateurs permettant d’effectuer les opérations algébriques de base entre deux instances : +=, -= et \*= sont définis à l’intérieur de la classe puisqu’ils nécessitent de manipuler directement les attributs des instances sur lesquels ils sont utilisés tandis que les opérateurs +, - et \* (addition, soustraction et multiplication directes entre deux vecteurs) peuvent être définis en externes puisqu’ils utilisent les opérateurs internes pour manipuler les attributs. L’opérateur =, défini en interne, permet l’affectation directe de nouvelles valeurs passées en arguments aux coordonnées du vecteur, tandis que ~ (interne également) permet de définir le vecteur comme vecteur unitaire.

*Semaine 5*

**[Question P6.1]** *A ce stade, quels sont les attributs, méthodes et constructeurs de votre classe ChampPotentiel ?*

Nous avons choisi comme attributs trois entiers non signés répertoriant le nombre de cubes que compte le champ selon chaque direction (x, y et z) ; nous avons pensé les initialiser comme un tableau fixe de trois entiers avant de réaliser qu’il était plus simple de les utiliser comme trois variables indépendantes. Afin de pouvoir calculer les dimensions de la boîte, nous avons rajouté un attribut représentant le pas séparant chaque cube. Enfin, le champ lui-même est représenté par un tableau tridimensionnel de potentiels dont la taille sera initialisée plus tard à l’aide des autres attributs de la classe. Cette conception permet de modéliser facilement les trois dimensions de notre système ; de plus, tous ces attributs sont définis en privés afin de préserver l’encapsulation de la classe.

Le constructeur de la classe permet d’initialiser les attributs de la classe représentant le nombre de cubes et le pas à l’aide de valeurs passées en arguments.

*Semaine 6*

**[Question P7.1]** *Dans votre conception, la méthode iteration() est-elle public, protected ou private ? Justifiez votre réponse.*

Puisque la méthode iteration() n’est utilisée que dans la méthode resolution() et n’a pas besoin d’être appelée directement par l’utilisateur, nous avons décidé de la définir en private afin de garantir une bonne encapsulation.

**[Question P7.2]** *Dans votre conception, comment/où est représenté le paramètre ε de l'équation (6) ?*

Le paramètre ε est un argument la méthode iteration() initialisé avec la valeur par défaut 0.1.

**[Question P8.1]** *Que vous inspire le paragraphe précédent ? Cela introduit-il d'autres classes ? Cela modifie-t-il votre conception précédente ? Si oui, comment et pourquoi ?*

A la suite de ces nouvelles instructions, nous avons décidé de créer une superclasse nommée Boite3D dont héritent les classes ChampsPotentiels et Ciel : en effet, toutes deux sont définies comme une boîte tridimensionnelle définie à l’aide des mêmes caractéristiques. Ainsi, cette nouvelle classe possédera comme attributs (privés) les dimensions de la boîte (nombre de cubes et pas) comme décrites pour ChampsPotentiels à la question P6.1 ainsi qu’un tableau tridimensionnel privé dont le type sera défini à l’aide de templates. Nous avons choisi d’utiliser des templates définies avec les types Potentiel et CubedAir afin d’optimiser l’héritage : ainsi, le tableau tridimensionnel d’instances des classes ChampsPotentiels ou Ciel s’initialisera des Potentiel ou des CubedAir respectivement en accord avec la classe de l’instance.

**[Question P8.2]** *Parmi les grandeurs citées ci-dessus, lesquelles représentez-vous sous forme d'attributs et lesquelles sous forme de méthodes ?*

Nous avons choisi de représenter toutes ces grandeurs sous forme de méthodes à l’exception de la vitesse du vent, qui est un attribut privé représenté par un tableau fixe de double à 3 éléments (valeur de la vitesse dans chaque dimension). Cette conception nous semblait la plus adaptée puisque l’enthalpie, la température, les différentes pressions et l’état sont des valeurs que l’utilisateur et le programmeur doivent pouvoir déterminer directement à n’importe quel point du système tandis que la vitesse ne nécessite pas d’appel direct ; elle nous permet également d’éviter d’effectuer ces calculs lorsque ce n’est pas nécessaire (pour remettre à jour l’état du CubedAir par exemple).

*Semaine 7*

**[Question P9.1]** *En termes de POO, quelle est donc la nature de la méthode dessine\_sur() ?*

Il s’agit d’une méthode virtuelle pure car sa définition dépend entièrement du type d'élément sur lequel elle est utilisée - une définition générale n'aurait donc pas de sens et serait terriblement contraignante. En effet, chaque classe héritant de Dessinable possède des attributs différents et nécessite donc d’être dessinée d’une manière très spécifique.

**[Question P9.2]** *Quelle est la bonne façon de le faire dans un cadre de programmation orientée-objet?*

On peut choisir de créer une collection de montagne grâce à un tableau vector. Mais cela a plusieurs inconvénients. Le premier étant de ne

pas pouvoir avoir différents types de montagnes. Il est donc préférable d'utiliser des pointeurs sur des montagnes qui pourront ainsi pointer sur des types de montagnes différents

**[Question P9.3]** *A quoi faut-il faire attention pour les classes contenant des pointeurs ? Quelles solutions peut-on envisager ?*

Il faut faire attention à ce que ces pointeurs existent aussi longtemps que la classe Systeme mais aussi qu'ils ne continuent pas d'exister après

En bref, il faut faire attention à la désallocation de mémoire. Lorsque que l'objet système est supprimé grâce au destructeur, si on ne fait rien,

les montagnes subsistent en mémoire et il n'y a plus aucun pointeur pour se « rappeler » son adresse. Il y aura fuite de mémoire. Pour éviter

cela, il faut faire un delete des montagnes dans le destructeur de Systeme afin que qu'elles soient supprimées avant.

**[Question P9.4]** *Comment représentez-vous la classe Systeme ? Expliquez votre conception (attributs, interface, ...).*

La classe Systeme représente ce qui compose le système physique à modéliser. Elle doit posséder donc des montagnes, un cie

*Semaine 10*

**[Question P12.1]** *Comment représentez-vous ces nouveaux objets « chaînes de montagnes » ? Où s'incrivent-ils dans votre conception ? Quelle(s) classe(s) avez-vous ajoutée(s) ? Cela change-t-il quelques choses aux montagnes que vous avez conçues jusqu'ici ?*

Afin de représenter les chaînes de montagnes, nous avons choisi de créer une nouvelle classe ChaineDeMontagnes héritant de Montagne (et de Dessinable par extension) – elle permet donc de d’implémenter le fait qu’une chaîne de montagnes est à la fois une montagne et une collection de montagnes. Cette nouvelle classe hérite donc d’une part des attributs (privés) de Montagne (coordonnées du centre, hauteur, étalement) mais possède également un tableau de Montagnes permettant de regrouper les montagnes qui constituent la chaîne.

Nous avons donc dû modifier la méthode altitude() de la classe Montagne afin de la rendre virtuelle : utilisée sur une chaîne de montagnes, elle permet désormais de retourner le maximum des altitudes de chacune des montagnes simples contenues dans la chaîne – elle-même calculée à l’aide de la définition originale de la méthode.