Héritage

Objectifs

L'objectif de ce TP est de développer un système de classes de manipuler différents modèles d'évolution de surface d'océan. Ces classes feront appel à la classe `Dvector` développée lors des premières séances. Cette classe pourra être utilisée par composition, agrégation ou dérivation.

Les notions abordées dans ce TP sont : l'héritage, l'encapsulation, la composition et l'agrégation.

Vous êtes libre de modifier la classe vecteur pour l'adapter aux besoins qui pourront apparaître : vous pouvez ajouter des opérateurs, des attributs ou des méthodes.

Lors de l'implémentation de ce nouvelles classes, vous devrez apporter un soin tout particulier à la gestion de la mémoire et à la documentation de votre code.

Les données des différentes classes doivent, dans la mesure du possible, être stockées de façon contiguë en mémoire.

La gestion d'erreurs doit se faire via des exceptions (utilisez `throw`).

Implémentation

1. La première classe dont nous avons besoin ici est `Height` qui contiendra l'ensemble des hauteurs calculées dans une boîte carrée de taille physique (L_x, L_y) discrétisée en (n_x, n_y) éléments. La classe `Height` devra proposer les fonctionnalités suivantes
   - Les 4 méthodes de base d'une classe en identifiant celles qui doivent être désactivées, c'est-à-dire en exposant uniquement celles utilisables par un autre programme.
   - Un opérateur `()` qui permettra de considérer la classe comme un foncteur. Le foncteur pourra prendre en paramètres la position horizontale (x, y) et le temps t.
   - Des accesseurs pour les différents paramètres.

   Certains paramètres ne doivent pas être modifiable pendant la simulation numérique.
2.

Remarque 1

Il peut être intéressant de simuler dans un premier temps les situations mono-dimensionnelle. Pour cela, on pourra prendre $n_y = 1$.

Vous pouvez modifier les opérateurs de flux afin de générer un fichier texte que l'on peut afficher avec le logiciel `Gnuplot`. Un tel fichier se compose de colonnes de coordonnées (x, y, z). A chaque changement de coordonnée y, on ajoute une ligne blanche.

3. Construire une classe `WaveModel` abstraite qui servira aux différents modèles qui seront développés. Cette classe ne contiendra un attribut pour la direction du vent, un attribut d'alignement moyen des vagues avec le vent, un attribut d'intensité, un paramètre de longueur d'onde moyen et enfin un paramètre d'ajustement de la hauteur des vagues pour simuler différents effets.

Ces éléments ne devront pas être modifiable une fois que la classe aura été instanciée, ni par la classe mère, ni par toute classe héritant de celle-ci. La classe `WaveModel` devra proposer les fonctionnalités suivantes :

- Les 4 méthodes de base d'une classe en identifiant celles qui doivent être désactivées, c'est-à-dire en exposant uniquement celles utilisables par un autre programme.
- Un opérateur `()` qui permettra de considérer la classe comme un foncteur. Le foncteur pourra prendre en paramètres la position horizontale (x, y) et le temps t.
- Des accesseurs pour les différents paramètres.

Chaque modèle ayant des structures et paramètres différents, la classe mère ne contient pas de méthode génératrice directement. Les classes filles possèderont donc un membre agrégé qui s'occupera de la génération.

4. Le premier modèle que nous allons implémenter est celui de Gerstner pour la houle. C'est un modèle qui permet de définir la houle comme étant une somme de fonctions trigonométriques, chacune de ces fonction étant une onde particulière.

   Avant de définir une dérivation de la classe `WaveModel`, nous devons donc définir une classe `GerstnerWave` qui définira chaque onde indépendamment. Comme préciser dans le document initial, une onde de Grestner se définie par les paramètres d'amplitude, de phase, de direction et de fréquence. La relation entre la fréquence et la direction est précisée dans le document descriptif du problème.

   Pour générer une direction cohérente entre les différentes ondes de Grestner, on fera l'hypothèse que toutes les vagues ont une direction proche de la direction moyenne du vent (ce sont les paramètres de la classe `WaveModel`). Pour l'amplitude de chaque onde de Gerstner, on s'appliquera à ce que le ratio amplitude-longueur d'onde soit identique pour toutes les ondes. Enfin, en ce qui concerne la phase, nous pourrons la choisir constante pour l'ensemble des vagues.

   Pour plus de réalisme, il est indispensable que certains paramètres ne soient pas modifiable au cours de la simulation.

   Le classe `GerstnerWave` devra implémenter, a minima, les méthodes suivantes

   - Les 4 méthodes de base d'une classe en identifiant celles qui doivent être désactivées, c'est-à-dire en exposant uniquement celles utilisables par un autre programme.
   - Un attribut seed uniquement accessible par les accesseurs.
   - Un opérateur `()` qui renvoie la hauteur d'un point.

5. Afin d'utiliser ce modèle depuis la classe `WaveModel`, il est nécessaire de dériver cette classe. Faire hériter une classe `GerstnerWaveModel` depuis cette classe mère. La classe `GerstnerWaveModel` devra contenir la liste des ondes de Gerstner.

> **Remarque 2**
>
> Dans ce système de classe, il manque encore une classe `Ocean` qui permettra de structurer les interactions entre les différentes classes mentionnées, une stratégie d'initialisation des données et la partie visualisation finale. Ces éléments feront partis des prochains TPs.

Analyse

Question 1

Pour chacune des classes implémentées, définir des tests pertinents.

Question 2

Justifier de manière précise les relations entres les différentes classes de votre implémentation.

Question 3

Tous les paramètres données permettent de simuler des vagues réalistes. Identifier les paramètres qui donnent des vagues plates et ceux pour lesquelles la houle est importante.