Module ITC313 - Informatique

Partie C / C++

TP9 + TP10 Héritage multiple et modélisation

Benoît Darties - benoit.darties@u-bourgogne.fr
Université de Bourgogne

Année universitaire 2016-2017

Exercice 1 : *Organisation d'un jeu de combat au tour par tour*

L'objectif de cet exercice est de mettre en évidence les différents mécanismes d'héritage vus en cours, naturellement l'héritage simple, mais également l'héritage multiple qui est possible en C++ mais pas directement en java.

Nous souhaitons mettre en place d'un jeu de combat au tour par tour. Un combat oppose deux combattants. A chaque tour, un combattant attaque avec un score donné son adversaire. L'adversaire se défend, une partie des points d'attaques est absorbée par son armure, l'autre partie est encaissée et ses points de vie diminuent. Puis le défenseur devient l'attaquant, et le combat continue jusqu'à ce que les points de vies de l'un des combattants atteignent 0. On vous propose de mettre en place une série de classes modélisant les combattants ainsi que le système de combats. Nous utiliserons plusieurs types de combattants, notamment un `Guerrier`, un `Mage`, et un `MageGuerrier`. Chacun de ces combattants possède des attributs communs, et sont avant tout issus d'un même type `Personne`. Nous allons donc créer des classes pour chacun de ces types, en définissant les héritages adéquats.

1. **Modélisation** :
   (a) En vous basant uniquement sur la courte description du jeu ci-dessus et sur les identificateurs des types, quelle hiérarchie de classes proposeriez-vous pour modéliser les combattants de ce jeu ? Vous complèterez ce modèle tout au long de l'exercice en rajoutant les méthodes et attributs au fur à mesure, lorsque ces derniers sont évoqués dans l'énoncé.
2. **Création de la classe Personne :**
   (a) Dans un fichier `Personne.h` définissez la classe `Personne` avec les attributs et les méthodes suivantes. On ne demande que la définition des méthodes. La déclaration des méthode se fera de manière déportée, dans un fichier `Personne.cpp`.
      i. Un attribut `nom` de type `string` avec spécificateur d'accès `protected` ;
      ii. Deux attributs entiers nommés `pointsVie`, et `pointsDefense` avec spécificateur d'accès `protected` ;
      iii. Un constructeur dont les trois paramètre de type `string`, `int` et `int` initialisent respectivement les attributs `nom`, `pointsVie` et `pointsDefense`.
      iv. Un accesseur en lecture sur chacun de ces attributs.

v. Une méthode `recevoirCoup()` qui soustrait aux points de vie une valeur égale à celle de l'unique paramètre de la méthode ôtée de la valeur des points de défense, mais ne rajoute pas de points de vie si le nombre obtenu est négatif.

(b) Donnez la définition de chacune de ces méthodes et du constructeur dans un fichier séparé `Personne.cpp`

3. **Une première classe héritée, la classe `Guerrier`**

(a) Dans deux fichiers `Guerrier.h` et `Guerrier.cpp`, définissez une classe `Guerrier` héritant de `Personne` et qui comporte les éléments suivants :

i. un attribut entier `pointsAttaque` avec spécificateur d'accès `protected` ;

ii. un constructeur ayant 4 paramètres : les 3 premiers paramètres seront utilisés pour appeler le constructeur de la classe parent, le 4 sera dédié à l'initialisation de l'attribut `pointsAttaque`

iii. une méthode `attaquePhysique()` qui renvoie un entier aléatoire compris entre `pointsAttaque`/2 et `pointsAttaque` ;

4. **Une seconde classe héritée, la classe `Mage`**

(a) Dans deux fichiers `Mage.h` et `Mage.cpp`, définissez une classe `Mage` héritant de `Personne` et qui comporte les éléments suivants :

i. deux attributs entiers `magie` et `forceMagique` avec spécificateur d'accès `protected` ;

ii. un constructeur ayant 5 paramètres : les 3 premiers paramètres seront utilisés pour appeler le constructeur de la classe parent, les 2 autres seront dédiés à l'initialisation des attributs `magie` et `forceMagique`.

iii. une méthode `attaqueMagique()` qui renvoie un entier tiré de manière aléatoire, et compris entre `forceMagique`/2 et `forceMagique`, si et seulement si les points de magie (atribut `magie` sont supérieurs à 0. Auquel cas, la valeur de ce dernier attribut est décrémentée. Si les points de magies ateignent 0, la `méthodeAttaqueMagique()` renvoie 0.

5. **Héritage multiple et combats**

(a) Proposez un programme contenant une fonction `combat()` permettant de mettre en scène un combat entre deux personnages. Vous pouvez rajouter à vos classes l'ensemble des méthodes qui vous paraissent nécessaires, afin de proposer un système de jeu aussi portable que possible, permettant l'ajout de nouveaux types de combattants sans avoir à toucher une ligne de la fonction. Un combat implique deux personnes, et se déroule au tour par tour. A chaque tour, l'un des combattants lance une attaque, qui renvoie un nombre de points. Ces points sont passés en paramètre de la méthode `recevoirCoup()` de son adversaire. Puis l'adversaire contre-attaque, et ainsi de suite jusqu'à ce que les points de vie de l'un des duelistes atteigne 0.

(b) Rajoutez enfin un nouveau type de combattant décrit par la classe `MageGuerrier`, et héritant à la fois de `Mage` et de `Guerrier`. Résolvez toutes les ambigüités d'appels aux méthodes et aux constructeurs au moyen des outils vus en cours. Le mage-guerrier attaque tantôt avec la magie, tantôt avec une attaque physique. ici encore vous êtes libres d'ajouter les éléments nécessaires à la réalisation de cette classe.

Remarques / questions : benoit.darties@u-bourgogne.fr