Date : 10/09/2024

Séance 2 – Programmation orientée objet

Objectifs :

- Rappel des concepts de la programmation orientée objet (POO) en langage C# : classes, attributs et méthodes
- Mise en pratique des concepts de POO pour le codage d'un jeu de tour par tour.

Concepts :

- **Classe** : ensemble des méthodes et des attributs qui modélisent une entité manipulée par une application. Une entité représente un élément spécifique d'une application. Par exemple, un personnage, un élément de la scène ou même des fonctionnalités pour assurer la connexion réseau d'un jeu.

Par exemple,

```
public class Personnage {
      //attributs
      //méthodes
}
```

- **Attributs** : données internes à une classe. Autrement dit, ce sont des variables définies dans le contexte (ou portée) global d'une classe. Par exemple,

```
private     int         vie;
public      string      nomObjet;
public      void        afficheContenu(){…}
private     void        calculerVitesse (){…}
```

- **Méthodes** : fonctions définies dans le contexte d'une classe.

```
public void marche (){  //méthode
            Console.WriteLine (« Je marche »)
}
```

- **Modificateur d'accès** : mots-clés utilisés pour permettre ou interdire l'accès externe aux attributs et méthodes d'une classe.
  - ✓ **private** : attributs et méthodes accessibles seulement aux méthodes de la classe.
  - ✓ **public** : attributs et méthodes également accessibles à des entités extérieures à la classe.
- **Méthodes de types « get » et « set »** : méthodes utilisées pour contrôler l'accès externe aux attributs d'une classe.
  - Type « get » : ce type de méthode est utilisé pour renvoyer la valeur d'un attribut d'une classe à une entité extérieure, comme la fonction « main » ou à une méthode d'une autre classe. Vous devez coder une méthode « get » par attribut.
  - Type « set » : il permet à une entité extérieure à classe de changer la valeur d'un attribut. Vous devez coder une méthode « set » par attribut.

Attention : l'utilisation des méthodes « **get/set »** pour l'accès aux attributs d'une classe est fortement conseillée à la place de l'utilisation du modificateur d'accès « public ».

Vue de l'ensemble :

```
public class Personnage {
private int vie;        //attribut
private double vitesse; //attribut
public  int id;         //attribut
      public Personnage () { //constructeur
            vie = 100 ;
            vitesse = 5 ;
      }
      public void marche (){  //méthode
            Console.WriteLine (« Je marche ») ;
      }
      public void arrete (){ //méthode
            Console.WriteLine (« Je m'arrete ») ;
      }
      public int getVie(){ //méthode de type « get »
            return vie ; }
      public void setVie(int nouvelleValeur){ //méthode de type « set »
            vie = nouvelleValeur ;
      }
}
```

- **Instance** : objet créé à partir d'un modèle défini par une classe. Nous pouvons considérer une instance comme une variable d'un « type » défini par une classe.

Pour créer une instance d'une classe, vous devez utiliser le mot-clé **new**. Par exemple,

```
Personnage monPerso = new Personnage();
```

- **Accès** aux **méthodes** et **attributs** d'une classe : pour utiliser (ou accéder) aux méthodes et attributs d'une classe, vous devez utiliser **l'opérateur « . »**. Par exemple,

```
static void Main(string[] args)
{
      Personnage monPerso     = new Personnage();
      //appel de la méthode « getVie() »
      int pointsVie           = monPerson.getVie(); /
      //accès à l'attribut Personnage.id
      Console.WriteLine(« Identifiant : »  + monPerson.id) ;
      Console.WriteLine(« Vie actuelle : » + pointsVie) ;
      ////appel de la méthode « setVie(int) »
      monPerso.setVie( 50 );
}
```

Exercices :

1. Identifiez les attributs et les variables :

```
public class Boss {
      public      int pointsAttaque;                  // ?
      private     int id;                             // ?
      string      nom ;                               // ?
      int         vitesse;                            // ?
      public Boss () {
            pointsAttaque     = 200 ;
            int vitesse       = 5 ;                   // ?
            string code       = nom + id ;            // ?
            Console.WriteLine(code) ;
      }
}
```

2. Créez un projet sur *Visual Studio* et ajoutez le code source ci-dessous. Ensuite, changez les modificateurs d'accès de la classe « Ennemi » pour la rendre compatible avec le code source défini sur la « boucle du jeu vidéo ». Utilisez des méthodes « get/set » pour donner accès aux attributs de la classe.

Classe « Ennemi »

```
public class Ennemi {
      public      int         pointsAttaque;
      public      double      vitesse;
      private     int         id;
      public      int         vie;
      public Ennemi() {
            vie = 100 ;
            vitesse = 5 ;
            pointsAttaque = 200 ;
      }
      private void attaque(){ //méthode
            Console.WriteLine(« J'attaque : - » + pointsAttaque) ;
      }
      private void defend (){ //méthode
            Console.WriteLine(« Je me défend ») ;
      }
      private void incrementerPointAttaque(){ //méthode
            Console.WriteLine(« Je prépare mon attaque ! ») ;
            pointsAttaque++;
      }
}
```

Boucle du jeu vidéo :

```
static void Main(string[] args){
      Personnage monPerso = new Personnage();
      Ennemi      ennemi1 = new Ennemi() ;
      while(monPerso.getVie() > 0) {
            monPerso.marche();
            ennemi1.attaque() ;
            Console.WriteLine("Vie personnage :"      + monPerso.getVie());
            Console.WriteLine("Vie ennemi :"          + ennemi1.vie);
            Console.WriteLine("L'ennemi attaque :" + ennemi1.pointsAttaque);
            monPerso.setVie(monPerso.getVie() -20);
            ennemi1.incrementerPointAttaque() ;
      }
      Console.WriteLine("C'est fini. Game Over!");
}
```

3. Les diagrammes de classes permettent d'avoir une vision globale de l'ensemble des classes qui modélisent une application, leurs attributs (-), leurs méthodes (+) et leurs relations. Le diagramme ci-dessous représente les classes : *ennemi*, *personnage* et *allié*.

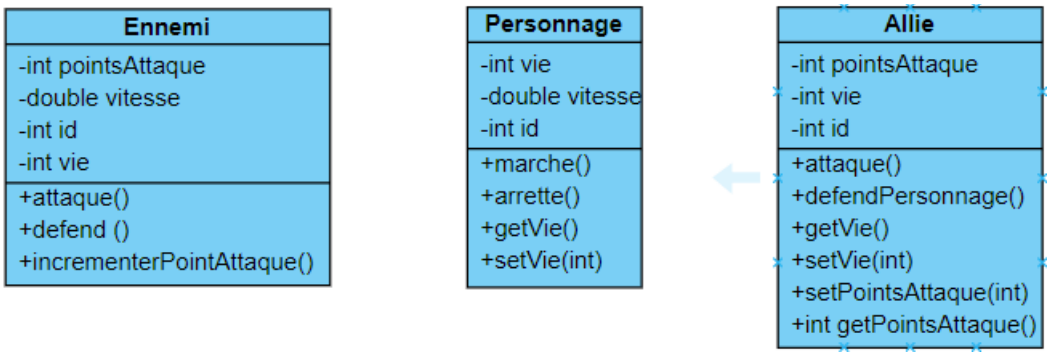

Figure 1. Diagrammes des classes du jeu exemple

- Ajoutez à votre projet les classes « Personnage » et « Allie » avec les méthodes et attributs définis dans le diagramme ci-dessus. Vous pouvez utiliser le code source des fonctions de même nom dans les classes « Ennemi » et « Personnage » comme exemple pour la nouvelle classe.
- Ajoutez une instance de la classe « Allie » au code de la boucle du jeu vidéo.
- Montrez le fonctionnement de la nouvelle classe par un appel à l'une de ses méthodes dans la boucle du jeu vidéo.

4. **Relier les points** : maintenant, nous allons faire interagir les objets créés à partir des classes : *Ennemi*, *Personnage* et *Allié*. L'objectif du jeu est que le personnage survive un nombre fini de tours du jeu. Le nombre de tours, N, sera un paramètre du jeu.
   1. Assurez-vous que votre jeu crée une instance de chaque classe.
   2. Codez la mécanique suivante pour le jeu :
      a) À chaque tour de boucle, le jeu demandera l'action à réaliser à l'utilisateur.
         - ✓ Touche « H » : on soigne le personnage, mais il subit les dégâts de l'attaque de l'ennemi au même temps.
         - ✓ Touche « Z » : on soigne l'allié et le personnage subis les dégâts de l'attaque de l'ennemi.

- ✓ Touche « D » : on demande à l'allié de faire le bouclier et de prendre les dommages causés par l'ennemi à la place du personnage. Le personnage se soigne eu même temps.

Attention, une fois que les points de vie de l'allié sont finis, cette action n'est plus disponible.

- ✓ Touche « A » : demande à l'allié d'attaquer l'ennemi. Le personnage subit les dommages infligés par l'ennemi.
- ✓ Autre touche : le personnage subit les dommages.

Astuce : la fonction « Console.ReadLine() » capture la touche appuyée par l'utilisateur.

b) Ensuite, l'ennemi essayera d'attaquer le personnage et le succès de son attaque dépendra de l'action choisie par l'utilisateur.
c) À la fin du tour, le jeu affichera les points de vie et d'attaque des éléments du jeu (instances de classes : ennemi, allié et personnage)
d) L'utilisateur gagne lorsque l'ennemi est vaincu ou que le nombre de tours est atteint. L'utilisateur perd une fois que la vie du personnage atteint zéro.

Perfectionnement :

5. Révisez votre projet et faites le nécessaire pour que la communication (ou liaison) entre les instances de classes soit faite par des méthodes afin d'éviter de donner accès direct aux valeurs des attributs d'un objet.
6. Paramétrez la valeur de **N** et les attributs des classes de façon à avoir un jeu cohérent. Par exemple :

Ennemi	Allié	Personnage
vie : 100	vie : 75	vie : 75
dommages d'attaque : 25	dommages d'attaque : 10	capacité de se soigner : 15

7. Pour les plus expérimentés, ajoutez une touche de hasard au jeu.
   - Pour chaque attaque (ennemi ou alliée), tirez un nombre au hasard pour déterminer si l'attaque réussit. Par exemple, si le nombre tiré est pair, les dommages seront égaux au maximum des points d'attaque de l'attaquant et zéro sinon.
   - Utilisez le nombre tiré au hasard (entre 0 et 100) comme base pour calculer les dommages infligés par une attaque. Par exemple, si le nombre 10 est sorti, l'attaquant causera des dommages équivalant à 10% de ses points d'attaque.

Astuce : Les commentaires nous permettent de décrire l'objectif et le fonctionnement de notre code. Un code source bien commentés facilite sa maintenance future. Deux options existent pour commenter votre programme :

a) /* bloc de texte en plusieurs lignes */
b) // texte sur une seule ligne.

Utilisez des commentaires pour décrire l'objectif et les paramètres de votre programme dans le code source.

Résumé :

À l'issue de cette séance, vous devez maitriser les notions suivantes :

- Les concepts de classe, méthode, attribut et instance.

- L’enjeu derrière l’interdiction/permission d’accès aux attributs et méthodes d’une classe et l’intérêt d’ajouter des méthodes de type « get » et « set » à une classe.
- Codage d’une classe à partir d’un diagramme.
- Création d’une application où des instances de différentes classes interagissent en utilisant leurs méthodes.