Examen - HMIN102 ("Ingénierie Logicielle") - M1 Informatique

Christophe Dony - Clémentine Nebut

Année 2016-2017, Session 1, janvier 2017.

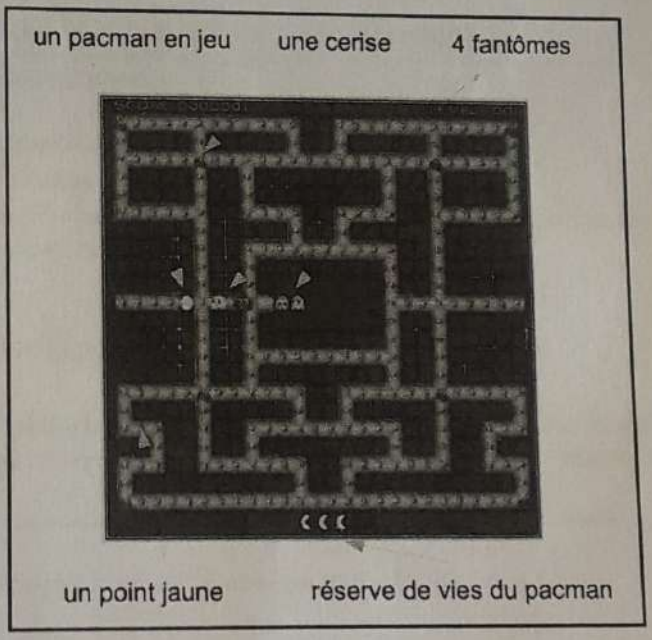
Durée : 2h00. Documents non autorisés. Il est possible de répondre aux questions de toute section sans avoir répondu aux autres ; mais la lecture dans l'ordre est vivement conseillée. Les codes donnés dans le texte sont écrits en Java, vos réponses peuvent utiliser un langage différent. Les explications doivent être précises et concises.

Contexte

Considérons comme contexte le jeu historique de type "PacMan". Les éléments d'un jeu sont un héros et les éléments que le héros peut rencontrer que nous nommerons les "*rencontrés*". Dans le jeu standard, il y a un seul type de héros, le *pacman*, et les rencontrés sont des *fantômes*, des *cerises* et des *points jaunes* (voir figure). Les *points jaunes* et les *cerises* sont fixes. Les *fantômes* sont mobiles ; leurs déplacements sont aléatoires. Le *pacman* est mobile, son déplacement est contrôlé par l'utilisateur (par exemple grâce aux touches "flèche" du clavier). Lorsque le *pacman* rencontre un *point jaune*, il le mange ; lorsque tous les points jaunes ont été mangés, le niveau courant est terminé et on passe au suivant. Lorsqu'il rencontre un *fantôme*, il perd une vie ; s'il n'a plus de vies, la partie est terminée. Lorsqu'il rencontre une *cerise*, il la mange et devient invincible par les fantômes pendant une certain laps (durée) de temps. Pendant ce laps de temps, une rencontre d'un *pacman* avec un *fantôme* envoie ce dernier en prison et rapporte des points supplémentaires. A la fin du laps de temps, le pacman reprend son comportement standard.

Nous nous intéressons à la réalisation informatique de ce jeu, et par extension à une réalisation d'un framework extensible permettant l'intégration aisée de variantes : nouveaux types de héros ou de "rencontrés". A cet effet nous imaginons les classes suivantes, le décalage à droite signifiant "est sous-classe de" (par ex. `Rencontré` est sous-classe de `Element`).

```
ElementJeu
    Héros
        Pacman
        AutreTypeDeHéros
    Rencontré
        Fantôme
        Cerise
        PointJaune
```



1 Framework version No1, Bases réutilisation.

1. Dans une version1 très simplifiée de l'implantation framework, on crée un nouveau jeu en créant une instance de la classe `Game` du listing 1.

   a) Question de cours : qu'est-ce qu'une affectation polymorphique ?

   b) Lors d'une exécution de la méthode `main` du listing 1, quelles sont les instructions qui sont ou qui induisent directement une affectation polymorphique ? Pourquoi ?

2. Dans une partie, on souhaite qu'il ne puisse y avoir au maximum que 4 instances de la classe `Fantome`. Si une tentative de création d'instance supplémentaire est tentée, une exception est signalée. Nommez et décrivez avec le code de la classe `Fantôme` une solution permettant d'implanter cette spécification (donnez uniquement la partie de code en lien avec la question).

```
public class Game {
    Héros héros;
    ArrayList<Rencontré> rencontrés = new ArrayList<Rencontré>();
    public void setHeros(Héros h) { héros = h;}
    public void addRencontrés(Rencontré r) {rencontrés.add(r);}

    public static void main(String[] args) {
       Game g = new Game(); //un jeu avec un pacman et deux fantôme
       Heros h = new Pacman(); Fantôme f = new Fantôme();
       g.setHeros(h);
       g.addRencontrés(f);} }
```

Listing 1 – classe Game de la version 1

3. Un pacman peut rencontrer lors de ses déplacements tous les autres éléments du jeu. La détection des rencontres est implantée par le moteur du jeu (dont le code ne vous est pas demandé). A chaque rencontre le moteur envoie un message `rencontrer(argument)` au héros courant (un *pacman* dans la version standard) avec comme argument l'objet rencontré, instance d'une sous-classe de la classe `Rencontré`. Dans cette première version du framework, on définit la méthode `rencontrer` sur Héros comme dans le listing 2. Cette méthode s'adapte à tout nouveau `Héros` ou `Rencontré` grâce aux envois du message `toString()`.

```
public abstract class Héros extends Element {
    public void rencontrer(Rencontré r){
        System.out.println( this.toString() + " a rencontré un " + r.toString() + "!"); }
```

Listing 2 – classe Héros de la version 1

a) Qu'est ce que le paramétrage (ou adaptation) par composition ?

b) Comment la méthode `rencontrer(Rencontré)` précédente est-elle paramétrée ?

c) En considérant la classe `Héros` comme une partie du Framework et les envois du message `toString` comme des points d'extension du framework, comment réaliser une extension ?

2 Framework version No2 - Spécialisations.

On oublie la version précédente de la classe `Héros` et de la méthode `rencontrer(Rencontré)` car on souhaite maintenant implanter le fait qu'on exécute un code différent pour chaque sorte de "rencontre". On réalise cette nouvelle version.

```
public abstract class Héros extends Element {
    public void rencontrer(Rencontré r) {System.out.println(''Suis-je utile?'');}
    public void rencontrer(Fantôme f);
    public abstract void rencontrer(Cerise c);
    ... }

public class Pacman extends Héros {
    ...
    public void rencontrer(Fantôme f){
        System.out.println("pacman rencontre un fantôme");}
    ... }
```

Listing 3 – classes Héros et Pacman de la version 2

Au niveau du moteur de jeu, on prend comme hypothèse que l'envoi du message `rencontrer(...)` est réalisé dans le contexte du listing 4.

1. Pour réaliser la disctinction des cas, pourquoi a-t-on défini plusieurs méthodes `Rencontrer(...)` et pas une seule utilisant l'opérateur *instanceOf*? Discutez les deux solutions en terme de réutilisation.
2. Quelle est la méthode invoquée par l'appel `h.rencontrer(f)` du listing 4 ?

```
1  Heros h = new Pacman();
2  Fantome f = new Fantôme() ;
3  while (jeuNonFini){
4    //à chaque fois que le pacman rencontre un fantôme
5    h.rencontrer(f);
6    ...}
```

Listing 4 – moteur de jeu de la version 2

3. Même question que 2b) en supposant que l'on n'ait pas défini la méthode "`public abstract void rencontrer(Fantôme f)`;" sur la classe `Héros`.
4. Expliquez vos réponses aux questions 2 et 3 précédentes en terme de redéfinitions en présence de typage statique.
5. En considérant le système comme une ligne de produit, imaginez un modèle de caractéristiques (*Feature Model*) permettant de générer des variantes du jeux. Représentez le modèle dans sa version graphique ou/et en utilisant la syntaxe de *FAMILIAR* avec le constructeur `FM`.

3 Gestion des changements de comportement (pour la version No 2).

En restant dans le contexte de la version 2 (listings 3 et 4)on s'intéresse au problème indépendant de la réalisation du changement de comportement du *pacman*, déclenché lorsqu'il rencontre une cerise et faisant que, pendant un certain laps de temps et avant de redevenir normal, il envoie les fantômes qu'il rencontre en prison.

1. Donnez les raisons qui font que le schéma *State* permet de traiter ce cahier des charges ?
2. Décrivez en UML (précis, associations, déclarations attributs et méthodes) une solution logicielle utilisant le schéma de conception *State* pour implanter ce cahier des charges.
3. Donnez les éléments clé (n'écrivez que les parties du code en rapport avec la question posée) du code correspondant. L'aspect "comptage du temps écoulé" ne fait pas partie de la question et pourra être représenté par des commentaires dans le code. Précisez, où, quand et comment sont réalisés les changements d'état.

4 Framework version No 3, une version réutilisable réaliste du moteur de jeu.

La version 2 précédente du framework pose un problème pour l'écriture du moteur de jeu, en fait il est en fait nécessaire de pouvoir stocker les objets rencontrés dans une variable de type `Rencontré`. On garde les classes `Héros` et ses sous-classes inchangées par rapport à la version 2 et on réalise la version du moteur de jeu du listing 5.

```
1  Heros h = new Pacman();
2  while (jeuNonFini){
3    Rencontré r = //... une méthode qui rend le prochain objet rencontré.
4                  // instance de Fantôme ou Cerise ou PointJaune ou ...
5    h.rencontrer(r);
6  }
```

Listing 5 – Moteur de jeux de la version 3

1. En supposant que la variable `r` référence une instance de `Fantôme`, pourquoi l'appel `h.renconter(r)` invoque-t-il la méthode `rencontrer(...)` de la classe `Héros` ?
2. Suite au problème indiqué par la question précédente, sans modifier la classe `Héros` ni ses sous-classes, proposez une modification du reste du système pour que l'ensemble fonctionne correctement avec tous les types de *Héros* et de *Rencontrés* présents (invoquer la bonne méthode sur la bonne classe de *Héros*). Il faut conserver cette variable `r` de type `Rencontré` dans le code du moteur de jeu mais vous pouvez modifier son code (listing 5) . Avez vous utilisé un schéma de programmation connu pour résoudre le problème ?