Génie Logiciel : Conception (INF 5153) – Examen Final Hiver 2019

- Durée : 3 heures (18h – 21h)
- Salle d'examen : Local PK - 1705

Directives Pédagogiques :

- Les seuls documents autorisés sont *(i)* la *cheatsheet* UML de la page web du cours et *(ii)* une feuille de note manuscrite (format *Letter, recto-verso, non photocopiée*) personnelle.
- Vos réponses sont à donner directement dans le cahier d'examen fourni. Vous pouvez utiliser toutes les pages du cahier (les pages blanches préférablement pour les diagrammes UML)
- La « propreté » de vos diagrammes (et de votre copie) fait partie des critères d'appréciations.
- Si le sujet vous parait ambiguë, expliquez dans votre copie le choix pris pour lever l'ambiguïté. Le surveillant ne répondra à aucune question
- Indiquez très clairement dans votre copie à quelle question vous répondez pour faciliter la correction.

Partie A : Des petites questions de cours (/20, ≈ 30 minutes)

Répondez en un paragraphe maximum aux questions suivantes.

1) Quels critères utiliseriez-vous pour choisir entre le patron **Stratégie** (*Strategy*) et le patron **État** (*State*) pour permettre à un objet de changer de comportement ? **/5**
2) Définissez l'anti-patron **Blob** (ou **Classe Dieu**), et expliquez en quoi il est préjudiciable dans une application orientée objet. **/5**
3) Expliquez le rôle des métriques dans le cycle de vie d'un logiciel. Quelles sont les limitations que peuvent rencontrer les développeurs en les utilisant ? **/5**
4) Expliquez le rôle des tests d'acceptations (p.ex., des scénarios *Cucumber*) dans l'activité de conception d'un logiciel. **/5**

Partie B : Un Système Domotique (/ 30, ≈ 60 minutes)

On s'intéresse ici au développement d'un système de gestion d'application domotique. Dans ce type de système, on définit des **`Capteurs`** (p.ex., un thermomètre, un capteur de luminosité) et des **`Actionneurs`** (p.ex., un interrupteur, un thermostat). Des **`Règles`** permettent de relier capteurs et actionneurs, en déclenchant l'exécution d'**`Actions`** sur les actionneurs suite à leur déclenchement par une **`Condition`**. Par exemple, si la température du thermomètre intérieur est inférieure à 21ºC (la condition, obtenue à partir d'un capteur), on déclenchera l'allumage du chauffage (l'action, en lien avec un actionneur), et on l'arrêtera dès qu'elle sera supérieure à 23ºC.

Question B.1 : Capteurs Adaptables (/15)

Les capteurs matériels sont des équipements fragiles et ont une durée de vie limitée. De plus, les gammes de produits ne sont pas suivie par les fournisseurs, et il n'est pas possible de remplacer du matériel défectueux à l'identique, il faut souvent racheter des éléments équivalents mais pas complètement identique. On souhaite découpler le système domotique de l'hétérogénéité des capteurs. Par exemple, un capteur de type « **`Rx6534`** » possède une méthode « **`read(): Integer`** » retournant un entier entre 0 et 1024 qu'il convient de convertir en degrés Celsius à l'aide d'une opération mathématique décrite dans la documentation du catpeur. Mais un capteur de type « **`TZW45f6`** » possède une méthode

« **`setUnit(char): void`** » qui permet de choisir entre degrés Celsius (**`setUnit('C')`**), Fahrenheit (**`setUnit('F')`**) ou Kelvin (**`setUnit('K')`**), et une méthode « **`getTemperature(): Integer`** » qui retourne la température dans l'unité préalablement demandée.

On va utiliser un patron **Adaptateur** (*Adapter*) pour régler ce problème, et définir le concept de « **`CapteurDeTemperature`** », qui exposera une méthode « **`temperatureEnCelsius(): Integer`** ».

1) **Utilisez un diagramme de séquence** pour mettre en évidence comment ce patron permettra d'interroger de manière uniforme un **`CapteurDeTemperature`**, indépendamment du fait que ce soit une instance de **`Rx6534`** ou de **`TZW45f6`**. Pensez-vous que le choix du patron **Adaptateur** est ici justifié ?

Question B.2 : Conditions Composites (/15)

Les conditions de déclenchement des règles peuvent être de simples comparaisons (p.ex., la température est inférieure à 21ºC), ou des compositions de conditions (p.ex., la température intérieure est inférieure à 21ºC **ET** la température extérieure est supérieure à 24ºC). On s'intéresse ici uniquement à des comparaisons numériques (<, =, >), et à des compositions de conditions booléennes (Et, Ou et Non). Les valeurs à comparer sont obtenu de l'abstraction **`CapteurDeTemperature`** définie précédemment (via la méthode « **`temperatureEnCelsius(): Integer`** »).

Pour une condition donnée, il est important de savoir si celle-ci est satisfaite, et on peut ainsi appeler une méthode « **`estSatisfaite(): Boolean`** » sur une instance de **`Condition`**. C'est ce qui permettra de déclencher (ou non) les actions sur les actionneurs. On fait ici le choix d'utiliser un patron **Composite** pour permettre de traiter de manière uniforme ce comportement les conditions simples et les conditions composées.

2) **Utilisez un diagramme de classe** pour montrer comment mettre en œuvre ce choix de conception dans l'architecture du système de domotique. Illustrez par des notes comment les concepts composites propagent à leurs éléments composés l'opération **`estSatisfaite()`**. Pensez-vous que le choix du patron **Composite** est ici justifié ?

Partie C : Les envahisseurs de l'espace (/ 50, ≈ 90 minutes)

On s'intéresse pour cette partie à la conception d'un jeu de type « *Space Invaders* ». Dans ce type de jeu, le joueur contrôle un vaisseau spatial situé en bas de l'écran, vers la gauche et vers la droite (en utilisant les flèches directionnelles). Il peut tirer un projectile en appuyant sur la barre espace. Si un projectile touche un ennemi, celui-ci est détruit. Les ennemis situés au front (c.à.d., sans autre ennemis devant eux) ont une probabilité de déclencher un tir, qui dépend du type d'ennemi. Dans la figure 1, il y a 3 types d'ennemis, représenté graphiquement de manière différente. Sur une même ligne, tous les ennemis sont identiques. Un joueur possède 3 vies (chaque tir reçu fait perdre une vie), et la partie est perdue quand le joueur n'a plus de vie, ou quand les

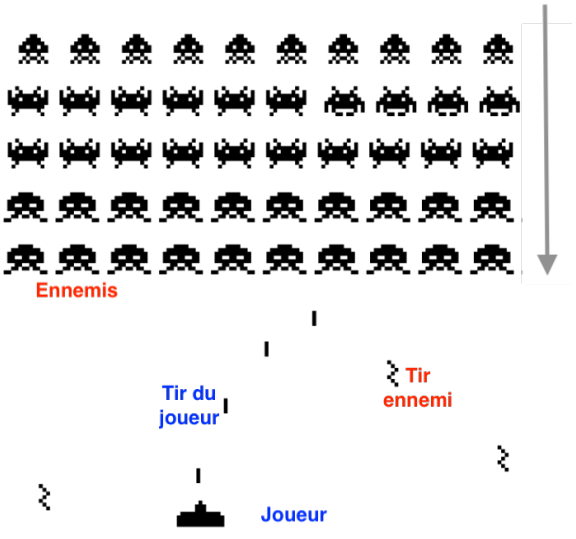

Figure 1. Capture d'écran du jeu "Space Invaders"

envahisseurs touchent le bord inferieur de l'écran. La partie est gagnée quand tous les envahisseurs ont été détruits.

Le jeu fonctionne au tour par tour, et à chaque tour on passe par les étapes suivantes :

- Le jeu écoute sur le clavier pour voir si une action a été demandée par le joueur ;
- Le jeu décide quel envahisseur va tirer ;
- Les actions sont résolues (les tirs ennemis et l'action du joueur sont appliqués) ;
- On vérifie les conditions de fin de partie. En l'absence de victoire ou de défaite :
    - Les tirs du joueur sont décalés d'un cran vers le haut,
    - Les tirs des envahisseurs sont décalés d'un cran vers le bas,
    - Tous les 10 tours, les envahisseurs sont décalés d'un cran vers le bas.

Pour chacune des questions suivantes,

- Utilisez le formalisme des diagrammes de classes et/ou de séquence UML pour illustrer vos choix de conception, selon ce que vous jugez le plus pertinent pour les mettre en valeur ;
- La cohérence de vos diagrammes est un critère de correction, ainsi que la justification de vos choix par rapport aux propriétés attendues. Ne paraphrasez pas les diagrammes, mais expliquez clairement les forces et faiblesses de vos choix de conceptions ;
- Précisez explicitement les principes de conception (parmi SOLID, GRASP et les patrons du GoF) que vous mettez en œuvre, ou au contraire ceux qui seraient de bons candidats mais pas forcément pertinent à date.

Question C.1 : Initialisation du jeu (/25)

Au lancement du jeu, il faut créer les rangées d'ennemis parmi les différents types d'ennemis existant dans le jeu. Le type d'ennemis utilisé dans une partie dépend du niveau de celle-ci (on commence au niveau 1, et chaque partie gagnée fait monter d'un cran la difficulté).

Votre conception doit respecter les propriétés suivantes :

- On doit pouvoir rajouter de nouveaux types d'ennemis, inconnus à date ;
- On doit pouvoir modifier facilement la manière dont le jeu est initialisé en fonction du niveau de la partie ;
- Seuls les ennemis du front (c.à.d., sans autres ennemis devant eux) peuvent tirer.

Question C.2 : Déroulement d'un tour de jeu (/25)

Pendant le déroulement du jeu, chaque envahisseur détruit rapporte des points (plus il est détruit tôt dans la partie et plus le nombre de point est important).

Votre conception doit respecter les propriétés suivantes :

- Le score est indépendant du type d'ennemi, et doit être mis à jour automatiquement ;
- Lors de la destruction d'un ennemi, celui qu'il masquait (si applicable) devient capable de tirer ;
- Le déroulement d'un tour de jeu est immuable, sauf le détail de chacune des étapes qui pourrait être modifiée dans le futur.