Sorbonne Universités

MASTER 1
PROJET CPS

Jeu : ***Dungeon Master***

Réalisé par :

Sofiane GHERSA - 3525755
Kahina FEKIR - 3711938

Année universitaire : 2017-2018

1. Introduction

Le monde d'informatique en général et de la programmation en particulier est très vaste, les informaticien font face à des très grand problèmes a chaque implémentation, parmi ces problèmes est celui de choix d'une architecture et de langage de programmation.

Il existe plusieurs architectures et langages et chaqu'un ayant des avantages et inconvénients.

Java est un langage de programmation orienté objet, la particularité et l'objectif central de ce dernier est que les logiciels écrits dans ce langage soient être très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, divers plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.

Dans la programmation orientée objet, la façon la plus classique d'ajouter des fonctionnalités à une classe est d'utiliser l'héritage. Pourtant il arrive parfois de vouloir ajouter des fonctionnalités à une classe sans utiliser l'héritage. En effet, si l'on hérite d'une classe la redéfinition d'une méthode peut entraîner l'ajout de nouveaux bugs. On peut aussi être réticent à l'idée que des méthodes de la classe mère soient appelées directement depuis notre nouvelle classe.

De plus, l'héritage doit être utilisé avec parcimonie. Car si on abuse de ce principe de la programmation orientée objet, on aboutit rapidement à un modèle complexe contenant un grand nombre de classes.

Un autre souci de l'héritage est l'ajout de fonctionnalités de façon statique. En effet, l'héritage de classe se définit lors de l'écriture du programme et ne peut être modifié après la compilation. Or, dans certains cas, on peut vouloir rajouter des fonctionnalités de façon dynamique.

D'une manière générale on constate que l'ajout de fonctionnalités dans un programme s'avère parfois délicat et complexe. Ce problème peut être résolu si le développeur a identifié, dès la conception, qu'une partie de l'application serait sujette à de fortes évolutions. Il peut alors faciliter ces modifications en utilisant le pattern Décorateur. La puissance de ce pattern qui permet d'ajouter (ou modifier) des fonctionnalités facilement provient de la combinaison de l'héritage et de la composition. Ainsi les problèmes cités ci-dessus ne se posent plus lors de l'utilisation de ce pattern.

2. Problématique

Ce travail consiste à réaliser un jeu de type ***Dungeon Master.***

On donne une spécification d'un jeu similaire a Dungeon Master et d'implémenter cette spécification selon la méthode ***Design-byContract.***

3. Conception

Nos choix techniques sont :

- L'application est réalisée suivant la méthode ***Design-byContract.***
- l'implémentation est écrit en ***JAVA.***
- l'interface graphique est développé en ***JAVA SWING.***
- la spécification : Ce rapport se concentre sur quelques spécifications qu'on juge intéressantes de notre travail, il vise donc à expliquer les points les plus importants, les autres parties sont disponibles sur ***GITHUB.***
  " https://github.com/GHERSASofiane/Dungeon-Master " .

3.1. Spécification

- **Map :** On a ajouté l'observateur ***Grille*** une classe qui contient la matrice, ce service est ***Cloneable*** pour des tests ultérieur).

- **EditMap :** Ce service raffine le service ***Map***, il permet aux joueurs de modifier la grille et de vérifier que cette modification correspond bien aux exigences et contraintes du jeu (service ***Cloneable*** aussi pour des tests ultérieurs).

- **Environment :** Ce service définit en plus la nature des cellules vu qu'il *extends* de ***Map***. Ajoute le contenu de la cellule (joueur, monstre, ……) et permet de redéfinir la méthode de fermeture de porte afin d'assurer de ne pas la fermer sur une cellule contenant un objet.

- **Mob :** Ce service définit l'objet mobile contenu dans la cellule sa nature, sa direction, permet à un objet de faire les différents mouvements (avancer, reculer, tourner (droite ou gauche) et faire un pas vers la droite ou la gauche)

- **Entity :** Ce service *extends* de **Mob,** il ajoute **Hp** qui définit le nombre de points de vie et la méthode **step()** qui sera définit les classes filles.

- **Cow :** Ce service défini une entité qui bouge de manière autonome et indépendante dans notre cas on considère que " **monstre** " est une instance de ce service étant donné leur comportement identique pour le moment, en cas d'ajout de fonctionnalité de combat on doit lui créer son propre service.

- **Player :** Ce service extend de **Entity** et défini le joueur, a l'instanciation on vérifie bien qu'on a instancier sur sur un player et la méthode **step()** exécute la commande de joueur et on voit la représentation des cases à côté de joueur si sa direction vers le Nord comme suite :

(3 , -1)	(3 , 0)	(3 , 1)
(2 , -1)	(2 , 0)	(2 , 1)
(1 , -1)	(1 , 0)	(1 , 1)
(0 , -1)	(0 , 0)	(0 , 1)
(-1 , -1)	(-1 , 0)	(-1 , 1)

- **Engine :** Ce service représente le coeur de jeu qui contient tous les entity de jeu et principalement le joueur, la méthode **step()** va s'occuper d'exécuter tout les **step()** des entity.

Donc pour faire fonctionner notre jeu il nous suffit de manipuler une instance de cette classe, ce qu'on va faire pour l'interface graphique.

3.2. Interface graphique

//TODO

3.3. Utilisation

//TODO

4. Test

Afin de tester les différents services implémentés nous utilisons les commandes suivantes sur le terminal :

- Map : ***ant runMapTest***
- EditMap : ***ant runMapTest***
- Environnement : ***ant runEnvironmentTest***
- Mob: ***ant runMobTest***
- Cow: ***ant runCowTest***
- Player: ***ant runPlayerTest***
- Engine : ***ant runEngineTest***

5. Amélioration

Le jeu réalisé pour le moment permet de créer une grille à partir des fichiers .TXT qui se trouvent dans le package ***Fille.*** Il couvrent toutes les contraintes de base définies, afin de se déplacer et arriver à la sortie tout en respectant les règles du jeu.

Afin d'améliorer ce dernier nous envisageons ajouter :

- Monstres et Combat.
- Trésor.
- La gestion de grille a partir d’une interface graphique.

5. Conclusion

Ce projet est une bonne occasion de mise en pratique des différentes connaissances acquises dans le cadre de module CPS, nous avons pu développer un jeu entièrement basé sur la méthode ***Design-byContract*** qui a été une véritable découverte pour nous.

Nous avons pu mettre en corrélation les aspects théoriques et pratiques, sans éclipser les difficultés que nous avons rencontrées dans ce projet notamment la

spécification incomplète, possèdent des incohérences qui nécessite certaines modifications et adaptations aux besoins et règles du jeu.

Pour Conclure ce projet s’est avéré très enrichissant, En plus de le respect des délais et le travail en équipe seront des aspects essentiel, nous avons effectivement acquis de nombreuses connaissances profondes de développement des composants. Nous avons acquis des compétences nouvelles particulièrement la décomposition d’un grand projet en composants et la résolutions de chacun séparément de l’autre.