ULB UNIVERSITÉ LIBRE DE BRUXELLES

ECOLE POLYTECHNIQUE DE BRUXELLES

Projet Info-H-200 - Rapport final

Conception d'un jeu de type *Hack and Slash* en Orienté Objet

BA2

Théo Lepoutte
Tomas Coen

14 mai 2017

Table des matières

Introduction

Dans le cadre du cours de Programmation Orientée Objet, il est demandé aux étudiants de réaliser un jeu de type *Hack and Slash* avec une interface graphique (GUI). Le gameplay de cette variété de jeu consiste à faire évoluer un personnage sur un plateau (e.g. un labyrinthe) rempli d'ennemis. Le héros progresse alors en vainquant ces monstres et en ramassant objets et trésors. Ce projet a pour but pédagogique de mettre en pratique l'ensemble des concepts de la programmation orientée objet (OO) vus lors du cours *ex cathedra* et lors des séances de travaux pratiques.

Ce rapport est divisé en trois parties. Tout d'abord, il décrit l'architecture OO du code ainsi que les Designs Patterns employés. Ensuite, les différentes fonctionnalités implémentées sont passées en revue. Enfin, le code est présenté sous forme de diagrammes UML : de classe et de séquence.

1 Description de l'architecture Orientée Objet

Le patron de conception "Modèle-Vue-Contrôleur" a été choisi pour ce projet, celui-ci permet d'organiser au mieux l'architecture du jeu.

Un deuxième patron appelé "Observateur" a également été utilisé. Ce dernier fait partie des patrons dits structuraux. Il permet de coupler des classes en préservant leur dépendance.

Ces deux patrons de conception sont détaillés ci-dessous.

1.1 Modèle-Vue-Contrôleur

En théorie, le MVC divise le programme en trois parties distinctes ayant chacune des rôles différents :

— Le Modèle :
a pour but de traiter et stocker les données du jeu. Il est le seul à pouvoir les modifier et est donc responsable de celles-ci. Lors d'une modification de la base de données, le modèle envoie un message à la vue en signalant le changement.

— La Vue :
constitue la partie graphique du programme. Elle permet l'interaction entre le jeu et l'utilisateur. Cela se fait en deux temps. Premièrement, elle présente les résultats envoyés par le modèle. Deuxièmement, elle reçoit les actions de l'utilisateur et le transmet au contrôleur sans aucun traitement.

— Le Contrôleur :
permet de faire le lien entre le modèle et la vue. Il a comme rôle de mettre à jour et de synchroniser la vue et le modèle en fonction des notifications qu'il reçoit. Il ne possède que des méthodes de gestion d'événements et ne peut donc modifier aucune donnée.

En appliquant cette architecture, les interactions avec l'utilisateur, l'affichage et la gestion des données sont simplifiées. Regardons de plus près la façon dont nous nous sommes appropriés ce patron de construction.

1.1.1 Modèle

Notre modèle contient 25 classes dont l'entièreté, à l'exception de *Game*, *Inventory* et des Observers, hérite de la classe mère (abstraite) *GameObject*. A l'étage inférieur, se retrouvent trois classes mères abstraites placées dans des sous-packages différents :

— *Tile* : les cases du plateau de jeu (murs, caisses, portes, piques, flaques de poison).

— *Item* : les objets ramassables déposés sur le plateau du jeu (potions, flèches, armes). Nous avons également fait hériter la classe Damages de Item. Cette classe ne sert qu'à afficher les cases sur lesquelles les dégâts sont attribués et n'est pas réellement un Item. Cependant, il était plus pratique de la considérer ainsi au niveau de l'affichage, cela ne nuit en rien au déroulement du jeu.

— *Character* : les différents personnages pouvant se déplacer et attaquer (joueur, araignée, fantôme et mage)

Cette structure permet l'utilisation de concept OO tels que l'héritage, le polymorphisme ou encore le transtypage.

1.1.2 Vue

Notre vue contient plusieurs classes permettant de gérer l'affichage :

— *Launcher* : affiche l'écran de titre du jeu et permet de sélectionner la taille du plateau.

— *Window* : affiche la fenêtre principale contenant le jeu et l'HUD.

— *Map* : permet de dessiner le jeu et de visualiser les actions réalisées.

— *HUD* : permet de dessiner les différentes variables du joueur : XP, Vie, Mana ainsi que les objets présents dans l'inventaire.

— *Content* : permet de découper les images fournies dans le dossier *ressources* afin de récupérer les textures nécessaires sans pour autant devoir charger une grande quantité de fichiers ce qui aurait pu nuire à la vitesse du jeu.

— *GameOverPanel* et *LauncherPanel* : la création de ces classes était nécessaire afin d'afficher des images dans le fond des fenêtres GameOver et Launcher.

Cette structure permet de rendre la paramétrisation de l'interface graphique plus aisée.

1.1.3 Contrôleur

Le contrôleur se charge uniquement de gérer les entrées clavier. Chaque fois que l'utilisateur interagit au moyen du clavier, il reçoit l'information et le transmet au modèle.

1.2 Observateurs

De nombreux observateurs ont été utilisés dans le jeu afin de faciliter certaines manipulations, en voici la liste :

— *Movable* : tous les objets pouvant se déplacer implémentent cette interface (mobs et flèches). Celle-ci permet de notifier les observateurs à chaque déplacement de l'objet. Par exemple, chaque fois qu'un ennemi bouge, une notification est envoyée à la classe Game qui va alors notifier la classe Map afin de redessiner le jeu pour que le déplacement soit visible à l'écran.

— *MapChange* : lorsque le joueur, qui a implémenté MapChange, se déplace sur l'une des portes, une notification est envoyée au jeu qui va alors créer un nouveau plateau et déplacer le joueur de l'autre côté du plateau.

— *Removable* : tous les objets pouvant être enlevés du plateau après avoir été détruits implémentent cette interface, qu'il s'agisse des murs ou des mobs.

— *OpenDoor* : lorsque les ennemis sont tous vaincus, une notification est envoyée aux portes afin que celles-ci s'ouvrent et permettent au joueur de passer.

— *Refresh* : cet observateur permet seulement d'envoyer une notification à la classe Game pour que celle-ci puisse à son tour notifier la classe Map de redessiner le plateau.

— *GameOver* : lorsque la vie du joueur atteint zéro, une notification est envoyée à la classe Game afin de mettre fin au jeu et d'ouvrir la fenêtre des scores.

2 Fonctionnalités implémentées

2.1 Carte et déplacement

Le mode d'enchaînement des cartes choisi est le mode statique : à chaque fois que le joueur tue tous les ennemis présents sur la carte et traverse une des quatre portes, une nouvelle carte est générée aléatoirement. Sur cette carte se trouvent :

— **Le joueur**

— **Des ennemis** : une fois battus, en plus d'incrémenter l'expérience du joueur, ils peuvent laisser tomber une récompense : une potion de vie, une potion de mana ou des flèches.

— **Des blocs incassables** : ils permettent de délimiter la zone où le joueur peut se déplacer.

— **Des blocs cassables** : ils permettent de créer une sorte de labyrinthe et les caisses peuvent laisser tomber une récompense lorsqu'elles sont détruites.

— **Des portes** : une fois tous les ennemis abattus, les quatre portes s'ouvrent donnant ainsi accès à une nouvelle carte.

— **Des piques** : elles blessent le joueur lorsqu'il se déplace dessus.

— **Des flaques de poison** : elles désorientent le joueur pendant une durée limitée lorsqu'il marche dessus.

2.2 Déroulement du jeu

Une fois le jeu lancé, le joueur peut choisir la taille du plateau désiré et prendre connaissance des différentes touches nécessaires via l'écran d'accueil. Après quoi, il débute la partie.

Le joueur se retrouve alors dans un premier plateau où il peut récolter armes (épée et arc à flèches), potions et flèches.

Ensuite, un second plateau du jeu est initialisé : les ennemis et les différents blocs sont placés aléatoirement sur le plateau.

Deux scénarios sont possibles : soit le joueur meurt avant d'avoir éliminé l'entièreté des ennemis présents sur le plateau, ce qui met fin à la partie et envoie l'utilisateur sur le menu des scores, soit il les tue tous, permettant l'ouverture des portes. Une fois franchies un nouveau plateau est généré avec un ennemi supplémentaire si le niveau du joueur a augmenté.

2.3 Personnages

Le jeu possède deux principaux types de personnages : le joueur et les mobs ou ennemis (araignées, fantômes et mages), chacun possédant des capacités propres. Les mobs se déplacent aléatoirement, jusqu'à ce que le joueur entre dans leur champ de vision. À cet instant, ils se dirigent vers lui. Ceux-ci se distinguent entre eux de par leur vitesse de déplacement, leur champs de vision et d'autres capacités propres.

- **Le joueur** : possède trois types d'attaques : corps à corps, à distance et de zone.
  - L'attaque au corps à corps nécessite la possession d'une épée et permet d'infliger des dégâts à l'ennemi ou au bloc cassable se trouvant sur la case en face du joueur.
  - L'attaque à distance nécessite la possession d'un arc à flèches et de flèches. Il permet d'infliger des dégâts à l'ennemi ou au bloc se trouvant en face du joueur, peu importe la distance.
  - L'attaque de zone nécessite du mana, trouvable dans des flacons, et permet d'infliger des dégâts sur toute la zone entourant le joueur.
- **Les araignées** : possèdent une vitesse de déplacement plus grande que les autres ennemis.
- **Les fantômes** : peuvent traverser tous les blocs à l'exception des portes et des blocs incassables et ont un champ de vision plus élevé mais leur vitesse de déplacement est assez lente.
- **Les mages** : en plus d'infliger des dégâts, les mages désorientent également le joueur.

2.4 Items

Lorsqu'un ennemi est tué, un item est déposé. Une fois récolté, celui-ci se retrouve dans l'inventaire du joueur. Ce dernier peut alors l'utiliser à tout moment. Il en existe de quatre types :

— **La potion de vie** : permet de soigner le joueur. Celui-ci sera automatiquement guéri de sa désorientation et s'il n'a pas atteint son maximum de vies, en récupérera une. Le nombre maximum de vies augmente en fonction du niveau du joueur. Celui-ci est cependant limité à huit.

— **La potion de mana** : permet de rajouter un point de mana à condition que la quantité maximale de mana ne soit pas atteinte. Cette quantité augmente en fonction du niveau du joueur et peut atteindre jusqu'à huit slots.

— **La potion d'invincibilité** : permet d'empêcher au joueur d'être blessé pendant dix secondes et soigne sa désorientation.

— **La flèche** : est nécessaire afin d'utiliser l'arc à flèches. Le joueur ne peut pas en posséder plus de vingt.

2.5 Autres petites fonctionnalités

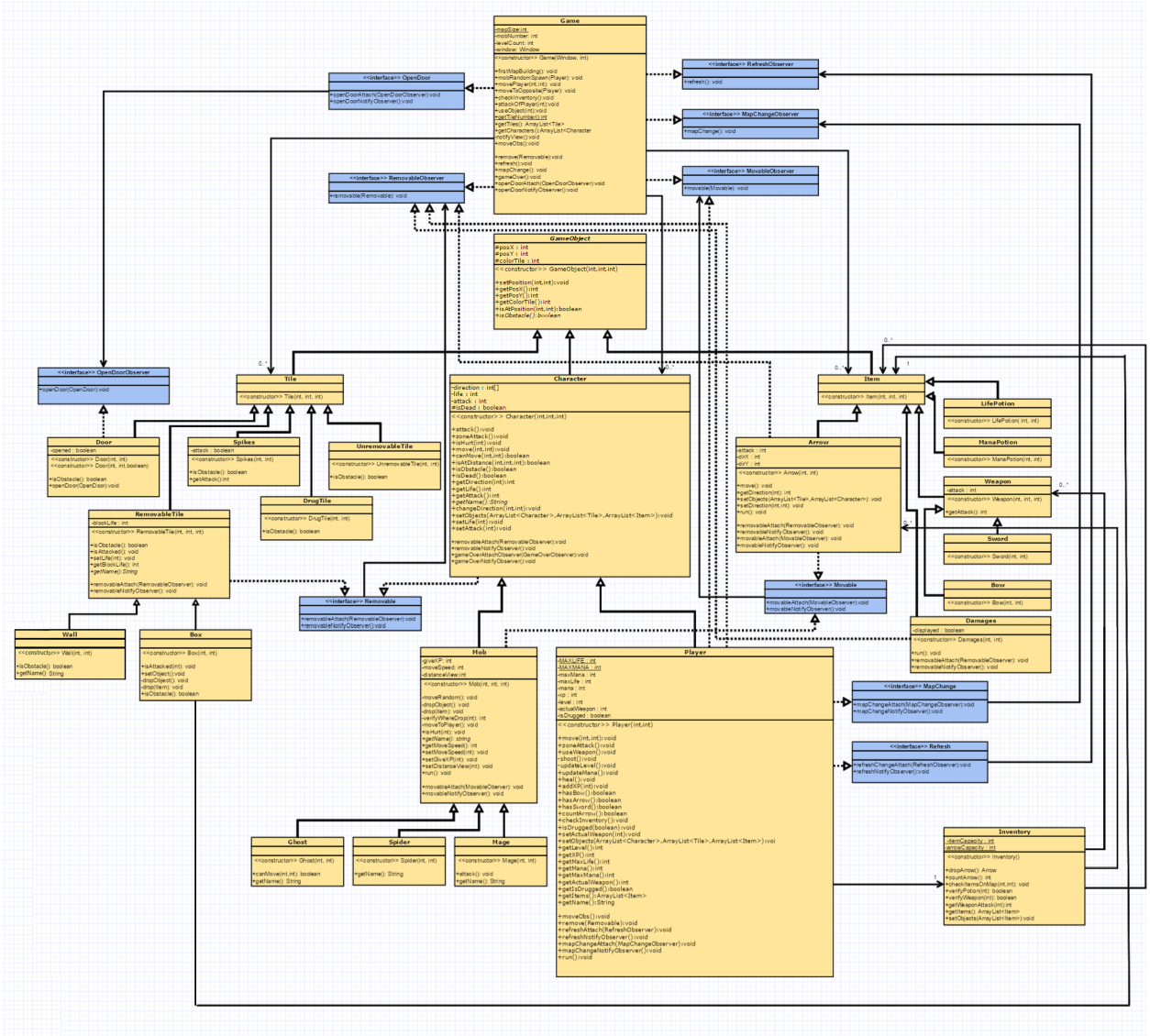
— **Intelligence artificielle** : les mobs, lorsque le personnage rentre dans leur champ de vision, vont chercher le chemin d'accès le plus rapide et qui, si possible, ne contient pas d'obstacle.

— **Drop des items** : lorsqu'un item est dropé, s'il se trouve sur une case inaccessible par le joueur, il va directement être déplacé sur une case vide.

— **Scoring** : les scores des joueurs sont enregistrés et les deux meilleurs scores sont affichés à la fin de la partie.

3 Diagrammes UML

3.1 Diagramme de classes



3.2 Diagrammes de séquence