RAPPORT
Méthode de conception
Licence 3 Informatique
Université de Caen Normandie

Siepka Aurélien 21906664
Pronost Sacha 21901956
Vallée Mathieu 21910887
Willenbucher Gurvan 21908377

Décembre 2021

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

Durant le début de ce semestre, il nous a été demandé dans la matière de méthode de conception de créer un jeu possédant différentes règles, tout en essayant d'implémenter au maximum les notions que nous avons vues durant les cours magistraux et travaux pratiques. Le jeu que nous devions implémenter était un "BomberMan" possédant les règles suivantes : possibilité de se déplacer, de poser une bombe ou une mine, de tirer, de déclencher un bouclier, et de ne rien faire. C'est donc en suivant ces différentes règles que nous avons développé notre application.

2 Présentation

Nous allons vous présenter dans la suite de ce rapport les grandes lignes de notre projet, en commençant par l'implémentation du modèle MVC (Model, View, Controller), notre bibliothèque implémenter pour représenter un personnage, et on finira par détailler notre utilisation d'un fichier XML afin de pouvoir configurer un personnage sans changer directement notre code.

2.1 Modèle MVC

Le modèle MVC nous a été imposé pour développer notre projet. Nous avons donc créé 3 packages : model, vue et controller. Le model sert de base pour le jeu, il constitue les données du plateau et des joueurs. Il n'y a ni besoin de la vue ni du controller pour que le joueur puisse jouer en mode terminal par exemple. La vue sert uniquement à afficher grâce à l'interface graphique (ici la librairie java swing) et ne modifie pas le model. Enfin le controller s'occupe de créer le model ainsi que la vue, et interagit entre les deux. Il s'occupe aussi de la gestion des actions (déplacement, actions diverses), de la souris.

2.2 Bibliothèque Personnage

Une bibliothèque spécifique a été créée et implémentée dans notre projet. Elle se nomme "PersonnageJeu" et elle a pour but de faciliter la création de personnage dans le cadre de notre jeu de stratégie. Cette bibliothèque contient donc une classe Personnage, représentant les données basiques d'un joueur (nom, énergie, munition, position), certaines de ces données sont récupérées par un fichier XML. Nous avons donc tout au long du projet utilisé cette bibliothèque pour implémenter nos joueurs.

2.3 Fichier de configuration XML

Afin d'attribuer la quantité d'énergie et le nombre de munitions du joueur de manière modifiable, nous avons décidé d'implémenter un fichier XML dans notre projet. Ce fichier contient une balise "personnages", une sous-balise "personnage", et dans cette balise se trouve deux sous balise "munition" et "énergie". C'est donc dans celle-ci que nous pouvons modifier les deux valeurs de notre personnage.

2.4 Architecture

Notre projet utilise le model MVC, il est constitué de trois packages principaux, model vue controller, ainsi qu'un package observer pour notre pattern observer.

Dans le package model se trouve les classes nécessaires au fonctionnement du jeu, on y trouve une interface Arme permettant de créer les Armes que peuvent poser les joueurs sur le plateau (dans notre cas, des bombes et des mines). La classe Case représente une case qui peut contenir un mur, une pastille d'énergie ou une Arme d'un joueur. Notre interface plateau permet de représenter un plateau, concrete Plateau est une implémentation de cette interface, et possède tous les éléments du jeu. Il est composé d'un tableau de Case, la liste de tous les joueurs et sa taille. Avec l'interface plateau et un concrete Plateau nous pouvons utiliser le pattern proxy pour faire une classe ProxyPlateau représentant le Plateau de jeu privé des éléments que le joueur n'a pas le droit de voir (bombe et mine). Une autre classe Action définie par des méthodes chaque action qu'un joueur peut effectuer. La classe model implémente toutes ces classes pour définir les fonctionnalités du jeu.

Le package vue possède toutes les classes permettant l'affichage des éléments du jeu en fonction du model. La classe Vue (héritant de JFrame) affiche la fenêtre principale et crée tous les autres composants à l'intérieur. VuePlateau (héritant de JPanel) affiche le plateau grâce à paintComponent de JPanel qui permet d'afficher des images. Ainsi, elle affiche chaque case, puis chaque élément (comme les murs, les bombes, les joueurs, etc ...). InfoJoueur (héritant de JPanel) permet d'afficher les informations du joueur, comme sa vie, son nom, ses munitions etc... ActionJoueur (héritant de JPanel) permet d'afficher tous les boutons des actions du joueur en fonction de son tour ou de ses attributs, par exemple s'il n'a plus assez d'énergie pour poser une bombe, le bouton pour poser une bombe ne s'affiche pas.

AdapterFromListeJoueursToTableModel implémente le pattern Adapter, il est détaillé plus tard. Image est la classe contenant toutes les images dans le type BufferedImage. Enfin, VueOptions permet d'afficher une fenêtre avec des pop-ups qui initialise la taille du plateau ou du nombre de joueurs en fonction de ce que l'utilisateur veut. Le model est créé grâce à cette classe via le controller.

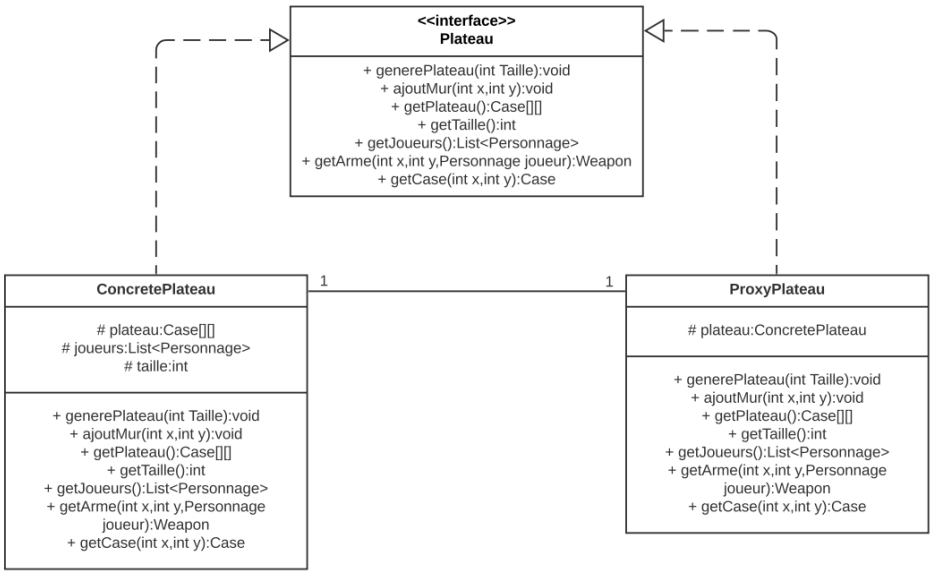
Enfin, le package "controller" permet de créer le model et toutes les vue. La classe Controller sert donc à créer un model et sa vue, et à lancer des actions en fonction de la vue. La classe EcouteurSouris (héritant de MouseAdapter de awt) permet, grâce à la méthode "mousePressed", de détecter un clic de souris et sa position par rapport à la case cliquée.

3 Design Pattern

Dans cette partie, nous allons vous présenter les différents patterns que nous avons implémentés dans notre projet. Chaque pattern sera explicitement détaillé, et contiendra un diagramme de classe afin de bien comprendre son fonctionnement dans notre programme.

3.1 Proxy

Une particularité du jeu était que chaque joueur avait la possibilité de placer une bombe ou une mine visible seulement par lui et non par les autres joueurs. Cette particularité nous a contraint à devoir utiliser le pattern Proxy afin de pouvoir afficher un plateau ne possédant que les mines et bombes que le joueur peut voir. Pour cela, nous avons donc implémenté trois classes différentes, une interface Plateau, une classe implémentant cette interface ConcretePlateau, et finalement une classe ProxyPlateau, implémentant l'interface Plateau, et prenant en argument un ConcretePlateau, qui décore donc la méthode "getArme", et envoie une arme si, et seulement si, elle est visible pour le joueur voulant la visualiser. Afin de bien visualiser et comprendre ce pattern, voici un diagramme UML de notre implémentation de proxy dans notre projet :



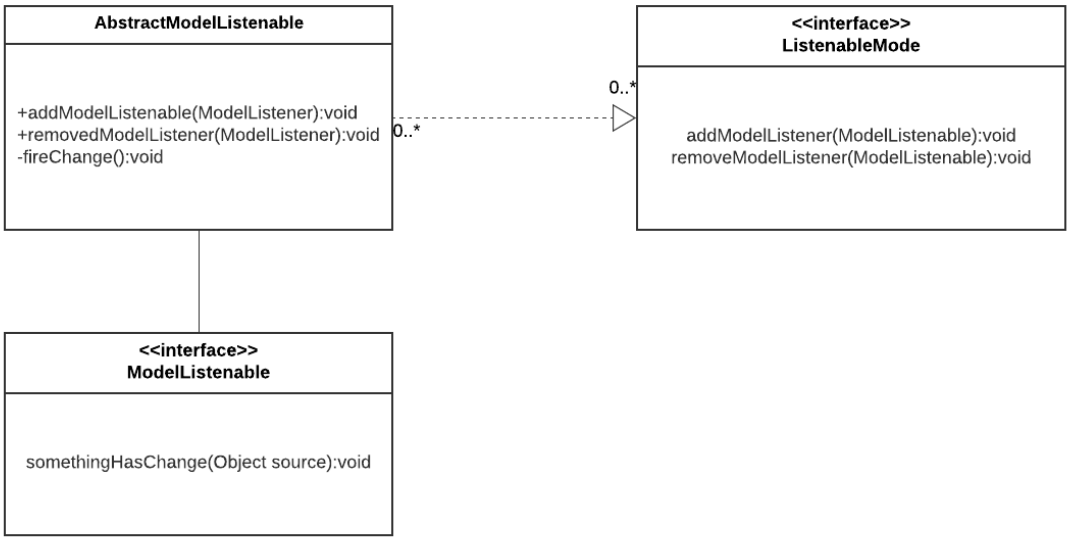
3.2 Adapter

Le pattern "Adapter" a été utilisé dans la vue afin d'adapter une liste de joueurs en tableau, pour pouvoir afficher la liste de tous les joueurs en fonction de leur nom ainsi que leur énergie. Voici le rendu :

ID	Nom	Énergie
1	Sacha	20
2	Aurélien	20
3	Gurvan	20
4	Mathieu	20

Grâce à cette classe (AdapterFromListeJoueursToTableModel) nous pouvons afficher ce tableau et l'actualiser quand quelque chose se passe dans le model.

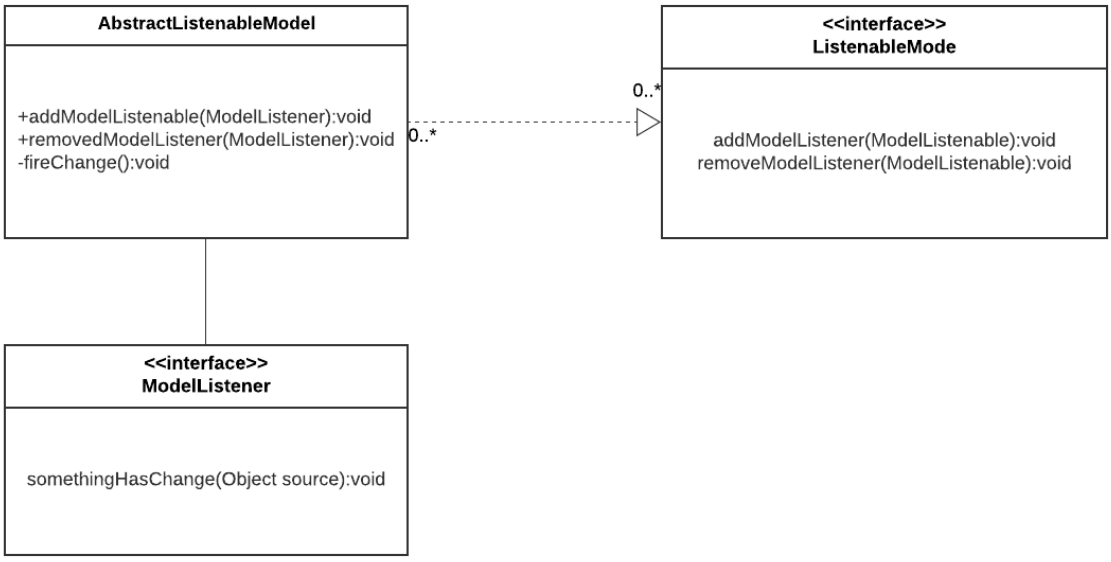
Pour ce faire, cette classe prend en argument dans son constructeur une liste de joueurs, et implémente les méthodes de l'interface TableModel de swing pour connaître le nombre de colonnes et de lignes, ainsi qu'avoir les valeurs dans chaque case du tableau. Le tableau est ensuite créé dans la vue grâce à la classe JTable de swing. Diagramme :



3.3 Observer

Le pattern "Observer" a été utilisé afin de permettre à la vue d'être actualisé si le model est modifié. Dans notre cas, fireChange() est appelé lorsque l'on change de joueur ou qu'une action est effectuée (les méthodes "changePlayer()" et "action()" dans Model). La vue implémente ModelListener, qui contient la méthode somethingHasChanged(), appelée grâce à fireChange(). Dans le controller, on fait Model.addModelListener(Vue) pour que la vue écoute le model.

Diagramme d'une des implémentations d'observer dans notre projet :



4 Conclusion

Pour conclure, certes notre projet n'est pas parfait, différent patterns en plus aurait pus être implémenté, notamment le pattern strategy dans notre modèle afin de rendre le code plus malléable, ou encore le pattern composite, afin que la méthode détonation ne se fasse que d'une traite. Cependant, nous sommes tout de même parvenus à implémenter différents patterns vue en cours, et nous avons appris de nouvelles choses pour la création d'un logiciel en groupe.