

Rapport de projet tutoré

Sommaire

| | |
|---|----------|
| Introduction | 3 |
| Étapes préliminaires et notre démarche | 4 |
| Analyse du sujet | 4 |
| Conception et modèle MVC | 4 |
| Un IDE les gouverner tous | 5 |
| Méthodologie et développement | 7 |
| Notre manière de procéder | 7 |
| Le développement | 8 |

Introduction

Pour terminer notre deuxième semestre, notre groupe a dû porter le jeu de société *Minivilles* sur PC. Ce projet a été réalisé dans le cadre du module M2107, en première année à l'IUT informatique du Havre.

Notre équipe, l'équipe 20, était composée de Valentin DULONG, Maxime MALGORN, Louis BOURSIER, Julien LE PÊCHEUR, et Richard BLONDEL.

Puisqu'il était difficile de respecter toutes les règles dans leur intégralité, le sujet nous proposait de coder une version légèrement modifiée de *Minivilles*.

Dans *Minivilles*, plusieurs joueurs construisent leur ville et la remplissent d'établissements. Pour bâtir un établissement dans sa ville, il y a un prix à payer, et c'est là qu'intervient la banque. Une autre manière d'engranger de l'argent, est d'avoir des bâtiments dans sa ville. Effectivement, à chaque lancé de dés, certains bâtiments déclenchent leur effet unique, en fonction du résultat du lancé.

Pour gagner la partie, il faut construire les quatre monuments de sa ville.

Plusieurs objectifs concernaient directement le produit fini. La première étape du projet demandait une interface en console, nous devions donc nous servir et perfectionner notre maîtrise du formatage de chaîne de caractères.

Pour la version réseau du jeu, il y avait également toute une partie qui concernait la gestion des erreurs. Un des scénarios pouvant être envisagé est la déconnexion accidentelle d'un joueur.

Il y avait aussi plusieurs objectifs pédagogiques. Tout d'abord, nous améliorer en conception orientée objet : nous avons choisi le modèle MVC (modèle-vues-contrôleur). Cela impliquait de bien l'appliquer, en créant une application modulaire, facile à maintenir et à modifier. Ce projet nous permettait aussi de mettre en oeuvre tout ce qui a été vu dans l'année : héritage, interfaces, classes abstraites ... Nous avons pu faire usage de ces mécanismes propres à la programmation orientée objet.

Dans les améliorations, il était aussi proposé de créer une version réseau du jeu. Il fallait alors créer un serveur et un client, ce qui allait nous pousser à nous améliorer en réseau.

Ensuite, et comme dans chaque projet, l'expression et la communication avaient une part importante. La rédaction de ce rapport en est l'un des résultats, mais nous devons aussi communiquer clairement entre nous : le travail à accomplir, et l'avancement des différentes tâches.

Finalement, on retrouvait aussi de la gestion de projet. Gérer ces différentes tâches avec les ressources à disposition impliquait de découper le travail, et de le répartir entre les différents membres du groupe. Un des objectifs pédagogiques visait donc à améliorer nos compétences organisationnelles.

Étapes préliminaires et notre démarche

Analyse du sujet

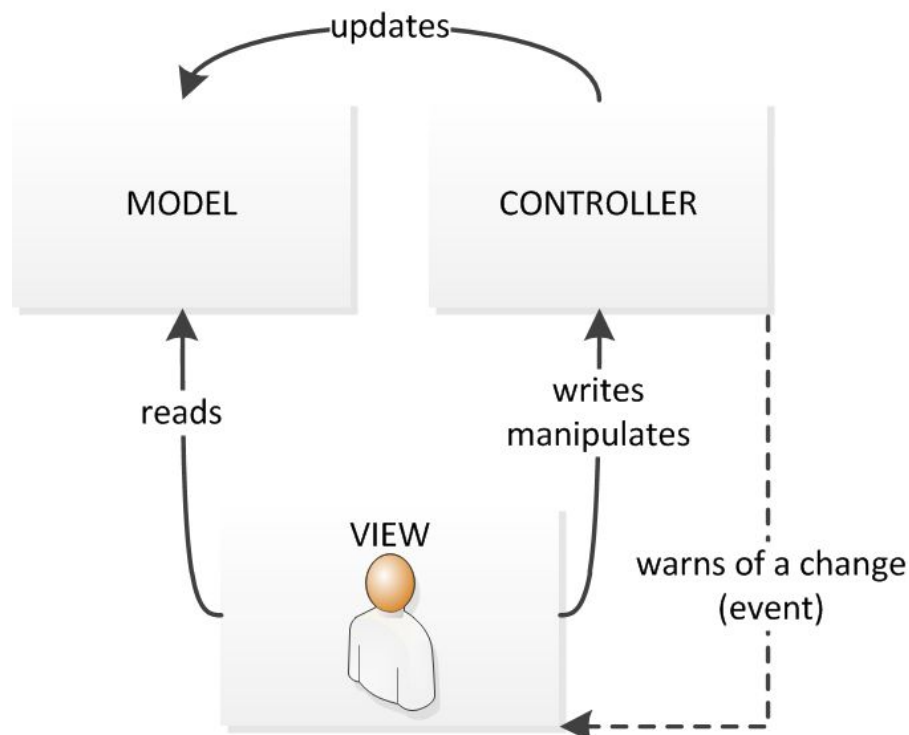
Avant de nous lancer dans le développement, nous avons dû lire et analyser le sujet. Bien comprendre l'application qui devait être réalisée était crucial pour faire les bons choix de conception par la suite.

Conception et modèle MVC

Tout d'abord, nous avons couché sur le papier quelques exemples de diagrammes de classes. Cela nous permettait de bien visualiser comment est-ce que le programme allait s'articuler.

INSÉRER DES EXEMPLES DE DIAGRAMMES QUE L'ON A ÉCRITS

Ensuite, nous avons choisi d'utiliser le modèle MVC (modèle-vues-contrôleur). Cela était le choix le plus pertinent, par rapport au jeu à développer. Effectivement, puisque toutes les cartes ont leurs effets, il y avait beaucoup de code logique à écrire dans la partie métier.



Interactions au sein du modèle MVC

Le modèle MVC permet aussi une bonne organisation du code, en séparant bien distinctement :

- la partie métier
- la partie interface, IHM
- le contrôleur

De cette manière, l'écriture du code fut claire et nous n'avions pas à nous demander où placer nos méthodes. La réflexion était rapide et nous pouvions nous concentrer sur la rédaction des méthodes.

Dernièrement, le modèle MVC permet aussi d'entretenir facilement le programme : il est aisé de mettre à jour une partie du code, ou d'ajouter une interface graphique pour la partie métier.

Un IDE les gouverner tous

Le langage de programmation choisi a été Java, puisque c'est le langage sur lequel nous avons travaillé toute l'année, dans le module de programmation orientée objet. De plus, programmer en un autre langage aurait impliqué un temps d'apprentissage non négligeable pour les membres de l'équipe.

IntelliJ IDEA a été le choix de prédilection de l'équipe. Il y a plusieurs raisons à cela. Effectivement, *IntelliJ IDEA* est un IDE, c'est-à-dire qu'il intègre tous les outils nécessaires au développement : éditeur de texte, compilateur, et console. Il est très modulaire et hautement personnalisable. C'est aussi l'IDE utilisé tous les jours par l'équipe en cours de TP et de TD. Enfin, il intègre *Git*, un logiciel de versions décentralisés.

Ce qui nous amène au point suivant : *Git*. *Git* a été notre choix pour synchroniser et relire notre code. Une fois le dépôt distant initialisé, nous n'avions qu'à mettre en ligne notre code sur ce dépôt à l'aide de *Git*.



La gestion des versions propose plusieurs avantages : nous pouvons revenir à une ancienne version du code, si nécessaire. Nous pouvons aussi travailler de chez nous et mettre en ligne le code, afin qu'il soit accessible aux autres membres du groupe. Enfin, *Github*, la plateforme qui hébergeait notre dépôt, propose une interface pratique qui permet, entre autres, de voir les changements apportés entre deux versions du programme.



```
Showing 2 changed files with 14 additions and 14 deletions. Unified Split
14 src/minivilles/metier/cartes/Cafe.java View
@@ -10,13 +10,13 @@ public Cafe () {
10 10
11 11     public void lancerEffet(Metier metier) {
12 12         int don = 1;
13 -         for(Joueur j : metier.getListeJoueur()) {
14 -             for (Carte c : j.getMain()) {
15 -                 if(c.getNom().equals("Café")) {
16 -                     metier.getJoueurCourant().retirerPiece(don);
17 -                     j.addPiece(don);
18 -                 }
19 -             }
13 +             if(this.getJoueur().getPieces() >= don) {
14 +                 metier.getJoueurCourant().retirerPiece(don);
15 +                 this.getJoueur().addPiece(don);
16 +             }
17 +             else {
18 +                 metier.getJoueurCourant().retirerPiece(metier.getJoueurCourant().getPieces());
19 +                 this.getJoueur().addPiece(metier.getJoueurCourant().getPieces());
20 20         }
21 21     }
22 22 }
```

*Visualisation des changements d'une version à une autre
sur Github*

La seule exception a été Valentin : n'étant pas habitué à *IntelliJ IDEA*, il a préféré utiliser *Gedit*, l'éditeur de texte par défaut installé sur les machines de l'université. En effet ce dernier lui est plus familier.

Méthodologie et développement

Notre manière de procéder

Une fois les deux étapes précédentes effectuées, nous pouvions commencer à travailler. Puisque nous étions 5, et que Julien n'a pas pu se déplacer à l'IUT, chaque membre du groupe a effectué divers travaux. Personne n'était réellement spécialisé en un domaine. Nous travaillions d'une manière similaire à une startup : une petite structure sans hiérarchie et où tous les membres effectuent des tâches variées.

Valentin s'est chargé de tenir un sommaire des tâches effectuées tous les jours.

Nous n'avons pas défini de point de contrôle, puisque nous passions la journée ensemble, nous savions à peu près quel était l'avancement du projet. Dernièrement, pour prendre connaissance des fonctionnalités implémentées par nos collègues, *Git* pouvait nous montrer ce qui avait été ajouté.

Une première version en ligne de commande

Pour débiter, nous avons naturellement suivi les étapes données dans le sujet. Une fois le dépôt *Git* initialisé et la structure du projet créée, nous avons commencé à écrire le code du jeu, en mode console.

Pour être certain d'avoir bien capturé les besoins, nous nous référons au livret des règles le plus possible.

La première étape a été d'implémenter la réserve, les mains des joueurs, et la banque. La majorité du temps a été utilisé pour faire une bonne interface en ligne de commande, lisible et agréable.

En effet, une des difficultés était d'afficher la situation actuelle du jeu comme dans le livret des règles. Les cartes devaient être affichées ligne par ligne, mais il fallait aussi afficher le solde de la banque, ainsi que les mains des joueurs.

Pour cette partie en console, beaucoup de tests ont été faits sur différentes machines, afin d'être certain que l'affichage était le même sur plusieurs tailles d'écran.

```
#####
# Joueur 2 #
# 1000 pièces #
#####
1. Champs de blé
2-3. Boulangerie
M1. Gare
M2. Centre commercial
M3. Parc d'attractions
M4. Tour radio
BILAN DU TOUR (J1)
Valeur du dé : 5
Pièces avant : 1000
Batiments activés :
aucun
Pièces après : 1000
MENU DE CONSTRUCTION (1000 PIÈCES)
1. Acheter une carte de la réserve
2. Construire un monument
3. Passer mon tour
choix : 1
```

Une fois la première étape terminée, le reste du travail consistait à implémenter les effets des cartes, le déroulement d'un tour de jeu, puis les monuments.

Plusieurs parties ont été jouées afin de vérifier que les effets de chacune des cartes fonctionnaient bien. Les corrections nécessaires ont pu être appliquées rapidement. Nous avons aussi ajouté les conditions de victoire : lorsqu'un joueur a construit les monuments de sa ville, la partie s'arrête.



Ensuite, contrairement à ce que l'on avait décidé lors de la phase de conception, Maxime a eu l'idée de regrouper toutes les cartes apportant des gains sous une classe mère *CarteGain*. Cela permettait de n'écrire le code donnant l'argent au joueur qu'une et une seule fois pour toutes les cartes.

De plus, le code est plus modulaire : si l'on devait coder l'extension de *Minivilles* mentionnée à la fin du manuel, il serait plus rapide d'ajouter certaines cartes.

Finalement, lorsque nous avons terminé cette version console du jeu, plusieurs choses ont changé par rapport à la conception prévue.

Par exemple, nous n'avons pas créé une classe fille par couleur de carte : rouge, vert, bleu, violet. De même, représenter les mains des joueurs face à face sur la console aurait été trop ardu, donc nous avons fait différemment.

Nous sommes ensuite passés à la version graphique du jeu.

La version graphique

Conception

Avant de nous lancer dans le développement, nous avons revu nos paquetages et leur organisation. Jusqu'à-là, nous avions cette structure :

- minivilles
 - ihm
 - IHM.java
 - metier
 - Metier.java
 - cartes
 - ...
 - monuments
 - ...
- Controleur.java

Afin de rendre modulaire l'interface de l'application, nous avons décidé de créer une classe abstraite nommée *IHM*. Dans cette classe abstraite, toutes les méthodes nécessaires à l'affichage sont données, sans code.

De cette manière, nous avons rajouté deux paquetages *console* et *gui*, dont les classes principales héritent de la classe abstraite *IHM*. Cela permet de changer à volonté d'interface.