Rapport Projet Java - Wargame

Ronan Abhamon, Florian Bigard, Nicolas Reynaud

2013-2014

Sommaire

1 Introduction

Le but du projet est de créer un jeu en interface graphique reprenant le thème du Seigneur des Anneaux.
Le plateau de jeu est découpé en cases, et chaque case peut être occupé par un soldat. Ce soldat appartient soit à l'armée des Monstres (les méchants), soit à l'armée des Héros (les gentils). Le but étant de détruire complètement tous les soldats de l'armée des monstres. Dans le cas contraire, le joueur a perdu.

Nous allons maintenant détailler la façon dont s'est déroulé le projet au sein de ce rapport.
Dans un premier temps nous ferons l'analyse du projet avec un schéma des différentes classes et interfaces crées.
Dans un deuxième temps, nous décrirons les techniques du langage orienté objet qu'est Java mis en œuvre dans le projet.
Nous continuerons alors en faisant une synthèse du résultat global du projet, en décrivant les différentes fonctionnalités implémentées.
Nous expliquerons ensuite comment le projet s'est déroulé et comment nous nous sommes organisé (au niveau du temps et du projet en général).
Puis nous citerons nos différentes sources (images et sons notamment) ainsi que les différentes documentations utilisées.
Enfin, nous terminerons par une conclusion qui fera un bilan du projet en général. Chaque membre du groupe en profitera pour donner son avis sur la programmation orienté objet.

2 Analyse du projet

En étudiant de plus près le sujet, plusieurs classes décrivant plusieurs éléments du projet étaient évidentes (car aussi donnés par le sujet) :

- **Pour les interfaces**
  - IConfig représentant la configuration du jeu (nombre de soldats, taille de la carte...)
  - ICarte qui donne la signature des méthodes que Carte doit implémenter
  - ISoldat, qui de même que pour ICarte doit donner la signature des méthodes que Soldat doit implémenter
  - CarteListener qui représente les événements de Carte
- **Pour les classe abstraites**
  - Soldat qui représente les méthodes et attributs communes aux monstres et aux héros
- **Pour les classes concrètes**
  - Aléatoire qui nous donnera des méthodes statiques pour générer des nombres aléatoires entre deux bornes
  - Carte qui représente la carte du jeu avec les actions associées
  - Heros et Monstre qui comprennent les méthodes spécifiques à ces deux

types d'objet
- Position permettant de connaître les positions des soldats sur la carte
- Historique permettant de conserver un historique des actions faites sur l'objet Carte (par exemple)

Nous avons ensuite convenu de faire une interface graphique plus poussée que celle demandée dans le sujet à l'aide d'images animées. Pour ce faire, il nous fallait créer plusieurs classe :

- Tileset : Par définition un tileset est une image composée d'un ensemble de tiles. Ces tiles sont des morceaux du tileset d'une taille de 32x32 ici. Le tileset permet donc de facilement créer une carte en sélectionnant une case de décor sur une grande image.
- Tile : Un tile est comme indiqué ci-dessus, est un morceau du tileset. Un tile est définit par sa praticabilité (Peut-on marcher dessus ?) et sa traversabilité (Une flèche peut-elle passer à travers ?). Il va de soit qu'avec cette configuration, il est inutile de passer par une classe Obstacle. D'autant plus qu'avec notre configuration, nous pouvons dessiner des grands décors, comme un arbre de 4 cases par 4.
- Charset : Une image contenant un ensemble de sprites d'un personnage. Un charset dans notre projet est composé de 4 directions :
  - Haut
  - Bas
  - Droite
  - Gauche

  Ainsi que de 4 animations de déplacements. Un charset est aussi caractérisé par une vitesse de déplacement et de visibilité sur la carte. Enfin Soldat hérite de cette classe.
- Infobulle permettant de dessiner une infobulle sur la Carte

Nous nous sommes aussi dis que des sons rendraient le jeu beaucoup plus plaisant et animé. Il nous fallait donc en plus créer une classe Son. Elle implémente des méthodes statiques permettant de jouer des bruitages, mais peut également être instanciée afin de gérer le son en arrière plan.

Nous avons créé le schéma UML des classes de notre projet, mais celui-ci étant trop imposant nous avons décidé de ne pas l'inclure dans notre rapport. Vous pouvez donc le trouver dans le même dossier que ce dernier.

3 Techniques

3.1 Environnement de travail

Nous codons sous le système d'exploitation GNU/Linux avec la version 7 d'OpenJDK.
Nous avons décidé d'utiliser Eclipse car c'est l'IDE dont nous avons appris à

nous servir en TP. De plus (quand on oublie ses nombreux crash, si on parvient à l'installer et si on suffisamment de temps devant nous pour l'exécuter), il fait bien son travail.

Concernant le partage des sources, nous avions déjà expérimenté l'outil de versionnage Git couplé à GitHub et cela avait très bien fonctionné. Nous avons donc décidé de réutiliser cette méthode malgré la menace qu'un autre groupe puisse nous plagier. Mais bon, il fallait d'abord qu'ils sachent que nous faisions notre projet sous GitHub, puis trouver nos pseudo. De plus, avec l'historique des commit nous pouvions faire preuve de notre bonne foi.

3.2 Classes Java utilisées

3.2.1 Classes graphiques

Nous avons décidé d'utiliser Swing pour notre interface graphique. Nous avons déjà décrit le fonctionnement dans la partie concernant l'analyse du projet.

3.2.2 Classes pour gérer le son

Pour gérer le son, nous avons dû plus ou moins ruser. En effet, dans les classes de base de Java permettant de gérer les son, rien ne permettait de jouer du MP3. Nous devions convertir les musiques trouvées en WAV. Hors, le format contenu par WAV n'est pas compressé. Les musiques de fond faisaient plus de 50 Mo. Ce n'était donc pas envisageable. Nous nous sommes alors tourné vers le format MIDI très léger. Il nous a donc fallu utiliser la classe Sequencer pour pouvoir jouer ce type de morceau.

De l'autre côté (les formats MIDI se faisant rare), pour les bruitages nous avons décidé d'utiliser du wav (car il s'agissait de séquences très courtes, le poids était négligeable). Il nous a alors aussi fallu utiliser la classe principale AudioClip.

3.3 Héritage

Nous avons décidé de faire hériter Monstre et Heros de Soldat car ces objets sont des soldats ; ils implémentaient beaucoup de méthodes et attributs communes. La classe Soldat hérite elle-même de l'interface ISoldat qui définit les méthodes que doit avoir un soldat. Soldat hérite aussi de Charset pour fixer son apparence.

Aussi, la plupart des classes héritent de IConfig car elles avaient besoin d'un élément de configuration.

Nous pouvons aussi citer Carte qui hérite d'un JPanel et la fenêtre principale qui hérite de JFrame.

Nous joignons avec le rapport le schéma UML de notre projet, intégrant les méthodes publiques et représentant les différents héritages. Nous n'avons malheureusement pas pu incorporer le schéma dans le rapport car la taille de celui-ci est trop importante

3.4 Polymorphisme

Soldat implémente les principales méthodes de Monstre et Heros. Les objets Soldats instanciés sont issus de l'une ou l'autre classe.

De plus, un objet Carte possède quelques méthodes prenant en paramètre un Soldat. Nous passons en paramètre de ce genre de méthode un Soldat rangé à la ième case d'un tableau de Soldat. Hors, nous plaçons dans ce tableau non pas des Soldats mais soit des Heros soit des Monstres.

3.5 Exceptions

Nous avons placé plusieurs gestions d'exceptions dans notre projet. Ainsi, nous vérifions que chaque image d'un type de Heros ou de Monstre est bien chargée. La classe Son implémente aussi quelques gestions d'exceptions. En effet, nous vérifions qu'un son midi est bien trouvé, que celui-ci est bien chargé et qu'il n'y a pas d'erreur lorsqu'on le joue. De même, pour les sons wav nous vérifions à l'aide des exceptions que le son est chargé correctement.

Nous créons de la même façon des exceptions lorsqu'on veut afficher le premier ou dernier message d'un historique vide, ou si nous tentons de déplacer un soldat hors de la carte.

3.6 Événements

Lors du déroulement du projet, nous nous sommes heurtés à un problème. Nous possédons un objet finTour implémenté dans FenetreJeu. Il nous fallait alors bloquer ce bouton lorsque les monstres exécutaient leur action. Hors, c'était Carte qui exécutait les actions des monstres, et il fallait donc que Carte puisse accéder au bouton de FenetreJeu. Nous avions plusieurs solutions :

- Passer le bouton en statique et donc pouvoir y accéder via Carte. C'était la pire solution possible, car pas POO du tout. C'était de la "triche", et vraiment pas adapté à une évolution possible du projet
- Passer FenetreJeu en paramètre du constructeur de Carte, pour que ce dernier puisse agir sur l'objet FenetreJeu via une simple méthode. Le soucis était donc que Carte et FenetreJeu étaient liés. Et une modification du bouton de FenetreJeu entraînait une modification alors une modification Carte.
- Passer par un événement. Lorsque Carte a fini de faire jouer les monstres, il déclenche un événement que FenetreJeu attrape. Lorsqu'il reçoit cet événement, il réactive tout simplement ce bouton. Cette méthode était à notre sens la meilleur solution. En effet, si nous décidions de ne plus avoir besoin de savoir quand les monstres avaient terminé leur tours, il nous suffisait pas exemple d'ignorer cet événement.

4 Synthèse

Le projet terminé, nous allons vous présenter différentes fonctionnalités que nous avons implémentées. Tout d'abord, nous avons décidé de créer une interface graphique. Les personnages sont animés quand ils se déplacent, mais pas quand ils combattent car nous n'avons pas pu trouver les images qui allaient.

Nous avons implémenté toutes les fonctionnalités demandées par le sujet. Nous allons donc présenter les principales, par ordre chronologique au lancement du jeu.

Lorsque le joueur lance le jeu, nous jouons une musique d'arrière plan choisie aléatoirement parmi trois autres musique. Le joueur peut alors lancer une nouvelle partie, ou charger une de ses parties sauvegardées. Il peut aussi couper le son via le menu ou un raccourci clavier.

Lorsqu'il est dans une partie, il possède l'historique des actions sur la partie bas-droite de la fenêtre. Cette partie n'affiche qu'une ligne de l'historique. Il est possible de remonter ou de descendre dans celui-ci en "scrollant" avec la souris. Le joueur peut aussi défiler dans l'historique via les boutons situés en haut de la fenêtre. Nous pouvons aussi avoir toutes les dernières actions effectuées en mettant la souris sur cette partie bas-droite. Une popup s'affichera alors sur la carte présentant l'historique complet.

Le joueur peut aussi sauvegarder une partie parmi 10 slots proposés via la sauvegarde rapide. Sinon, il peut choisir un endroit où sauvegarder sa partie via le bouton situé en haut de la fenêtre. Il en est de même avec le chargement.

Les héros sont sur la partie droite et les monstres sur la partie gauche. Ils sont tous placés aléatoirement sur la carte. Un brouillard est créé (et désactivable) en fonction des la position et de la portée des héros. Cela a pour but de cacher les ennemis hors de la zone des héros.

Le joueur peut déplacer, reposer ou faire combattre chacun de ses héros. Pour cela, il doit cliquer sur son héros. Nous avons aussi ajouté la possibilité pour le joueur de diriger ses héros au clavier. Il peut alors utiliser la touche TAB pour changer de héros sélectionné, puis les touches directionnelles pour déplacer son héros. Les cases où le joueur peut déplacer son héros sont en surbrillance. Lorsqu'il passe sa souris sur un soldat, une infobulle apparaît pour donner ses caractéristiques. Il est possible de reposer son héros en cliquant sur lui-même après l'avoir sélectionné, ou bien en ne déclenchant aucune action sur ce dernier à la fin du tour. Lorsque le joueur déplace ou fait combattre son soldat, des bruitages sont joués. En effet, si le soldat combat à plus d'une case d'un autre soldat, un bruit d'arc est exécuté. Si c'est du corps à corps c'est un bruit d'épée.

Si le joueur a fait jouer tous ses soldats, le tour est automatiquement terminé. Sinon, si le joueur clique sur le bouton fin de tour alors qu'il lui restait des héros à jouer, ces derniers se reposent automatiquement

Lorsqu'un soldat meurt, selon que c'est un monstre ou un héros un bruitage de cri est exécuté. De plus, il tourne sur lui-même avant de disparaître.

Concernant l'intelligence artificielle des monstre, l'algorithme est le suivant :

— Si un monstre possède peu de points de vie, il se repose.

— Sinon s'il voit un ennemi à sa porté il l'attaque.

— Sinon il se déplace.

Nous allons détailler un peu plus le déplacement d'un monstre. Lorsqu'un monstre attaque un héros, ce héros fait parti de cibles alors potentielles pour les monstres. C'est pourquoi lorsqu'un monstre doit se déplacer, si il n'y a aucun héros ciblés alors il se déplace aléatoirement. Sinon il se déplace vers une des cibles la plus proche de lui sauf si le chemin est bloqué (dans ce cas il se déplace là aussi aléatoirement). Cela a pour but de créer un attroupement de monstres vers les héros qui se font attaquer. Le joueur doit donc établir des stratégies, voir créer des "appâts" afin de triompher. Nous pensons aussi que ceci correspond au monde du Seigneur des Anneaux, les monstres se déplaçant souvent en groupe pour tuer.

Enfin, lorsque le joueur perd ou gagne, un son est joué et un message apparaît à l'écran. Il ne peut alors plus exécuter d'actions. Il faut logiquement qu'il recommence un partie, ou qu'il quitte le jeu.

Nous avons aussi rajouté le fait de pouvoir tricher. En actionnant le code Konami (haut haut bas bas gauche droite gauche droite puis entrée) il est possible d'avoir un menu spécial. Via ce dernier, on peut :

— Gagner automatiquement la partie
— Perdre automatiquement la partie
— Tuer n'importe quel soldat sur lequel on clique
— Ajouter un héros sur la case libre sur laquelle on clique
— Ajouter un monstre sur la case libre sur laquelle on clique
— Mettre tous les points de vie de tous les soldats à 1

5 Organisation

Au commencement du projet Ronan a décidé de créer la base de l'interface de la carte. En effet, il avait déjà des expériences en 2D et animations. Il s'est donc occupé de chercher les images adaptées et de créer les classes correspondantes.

Florian s'est alors chargé de créer la classe Son et de chercher les bruitages et musiques ainsi que de l'implémentation de la classe Position.

La carte créée, Nicolas pouvait créer une classe permettant d'afficher une sorte d'infobulle lorsque la souris passe sur un soldat. Via cette classe, des sortes d'infobulles qui montaient ou descendraient pour à la fin disparaître afin d'afficher l'augmentation des points de vies lors d'un repos ou la diminution de ces derniers après un combat par exemple. Il a aussi tenu à créer un menu de triche activable via le code Konami.

En ce qui concerne le reste du projet, tout le monde modifiait tous les fichiers après que l'on se soit convenu des tâches de chacun. C'est ici que nous nous sommes encore rendu compte de l'avantage d'utiliser Git, dans le sens où Carte était un peu le "noyau" du projet et tout le monde devait au moins légèrement modifier cette classe afin d'incorporer les nouvelles fonctionnalités.

6 Ressources

Nous avons cherché toute documentation et informations concernant le Java sur le cours bien évidemment, mais aussi sur Internet. Nous avons trouvé quelques pistes lorsque nous étions bloqué sur plusieurs forums comme StackOverflow, le SiteDuZero.

En ce qui concerne les sons, nous avons trouvé les bruitages sur soundfishing.net et les musiques MIDI sur moviethemes.net

Enfin, en ce qui concerne les images nous avons extrait les décors (herbe, rochers...) et quelques personnages du jeu The Legend Zelda - Minish Cap. Nous les avons assemblés à partir du jeu original (Par émulateur.) et avec des sources du site spriters-ressources

7 Conclusion

7.1 Analyse critique

Dans cette conclusion, nous allons chacun notre tour donner notre impression sur le projet, le Java et la POO en général.

Nous trouvons que notre interface graphique constitue le point fort de notre projet. En effet, nous l'avons beaucoup travaillé. Nous avons essayé de faire quelque chose de joli, en 2D avec des "effets" lorsqu'il y a des actions.

Nous trouvons que le point faible du projet peut-être l'intelligence artificielle, même si d'un point de vu logique les monstres du Seigneur des Anneaux sont pas bien intelligents.

Nous aurions voulu créer une fenêtre d'options permettant au joueur de configurer le jeu comme il le souhaite (nombre de monstres, taille de la carte...). Malheureusement nous nous n'avions pas le temps libre pour implémenter cette fonctionnalité.

7.2 Analyses personnelles

7.2.1 Abhamon Ronan

Projet sympathique. Cependant très insatisfait d'Eclipse, beaucoup d'instabilité sur de nombreux systèmes Unix. Soit je suis maudit, soit c'est réellement le cas. Mise à part ces problèmes, j'ai trouvé une très grande productivité avec Swing. Même si je vois Java ainsi que Swing comme une grande usine à gaz, je peux dire qu'en temps de développement c'est assez rentable.

Concernant la POO, je possède des notions par le C++ et Ruby. Java instaure cependant des concepts que je ne connaissais pas vraiment, j'ai donc pu comprendre plusieurs nouvelles choses par ce projet. Enfin, un grand avantage de la POO dans la conception de ce jeu est de facilement pouvoir décomposer le travail entre chaque personne du groupe. C'est un bon point sachant qu'un développement en C ou en d'autres langages plus bas niveau, il n'est pas évident

de s'organiser correctement. Ici il n'est pas nécessaire d'être très rigoureux pour avancer.

7.2.2 Bigard Florian

J'ai trouvé ce projet très intéressant. Faire un jeu est assez plaisant, surtout qu'il s'agit d'un domaine que j'aime bien (les jeux de stratégie et le Seigneur des Anneaux). Je connaissais déjà la POO avant ce cours, mais pas aussi poussé et j'avais beaucoup moins la logique POO.

Au final je peux affirmer que la POO est un concept qui me plaît beaucoup, car il est bien plus facile de séparer le travail lorsque nous travaillons en groupe. De plus, je trouve que le programme final est beaucoup plus facile à faire évoluer ou à corriger si on considère qu'il a été bien conçu au départ.

7.2.3 Reynaud Nicolas

Pour ma part, le choix du sujet et donc le projet était très intéressant. Ceci nous a permis de voir des aspects peu ou pas vue en cours comme par exemple de créer une file de message. Qui plus est, j'ai trouvé que coder sous Eclipse était vraiment très pratique et m'a permis de coder plus rapidement. Bien que celui-ci ne soit pas très stable et fort coûteux en ressource.

Ensuite, j'ai également apprécié la "liberté" que nous avions sur le projet. Ceci m'a permis d'inclure une chose que j'apprécie énormément : les codes triche (que je vous invite a découvrir et à tester). Mais cela m'a également permis de sans cesse chercher des améliorations puis de les implémenter (comme l'animation de gain/perte de point de vie).

En somme, j'ai réellement apprécié ce sujet tant par l'aspect technique (coder en Java et de façon assez rapide pour chaque module), que sur l'aspect relationnel (codage en groupe).