Rapport de projet

POO: Wargame

Groupe:

- BRUYERE Dimitri
- JEAMME Christopher
- PEURIERE Romain

Introduction

Le projet comme présenté par le sujet nous a demandé de créer un jeu de wargame en Java (Basé sur une guerre dans le monde imaginaire du seigneur des anneaux).

Celui-ci tout d'abord devait implémenter le fonctionnement de base demandé par le sujet et par la suite des améliorations libres (dont certaines étaient proposées).

Il s'agit donc d'un jeu tour par tour d'un joueur contre le général-ordinateur, le but étant à chaque tour de donner un ordre à ses différentes unités et de par une stratégie, réussir à éliminer tous les ennemis avant que ceux-ci n'y parviennent par inversement.

Tout ceci en utilisant à bon escient les principes de la programmation orientée objet (Java) que nous avons pu étudier en cours.

Nous allons donc présenter en plusieurs parties la manière dont nous sommes arrivés à ce résultat.

Plan

- Analyse du projet

  Cette partie décrira la structure de notre code par le biais d'un diagramme UML ainsi que l'explication de celui-ci.

- Techniques de POO mises en œuvre

  Nous décrirons ici quelques principes de la Programmation Orientée Objet nous avons utilisé.

- Présentation des résultats

  Il sera ici expliqué nos principales fonctionnalités.

- Temps de travail et répartition

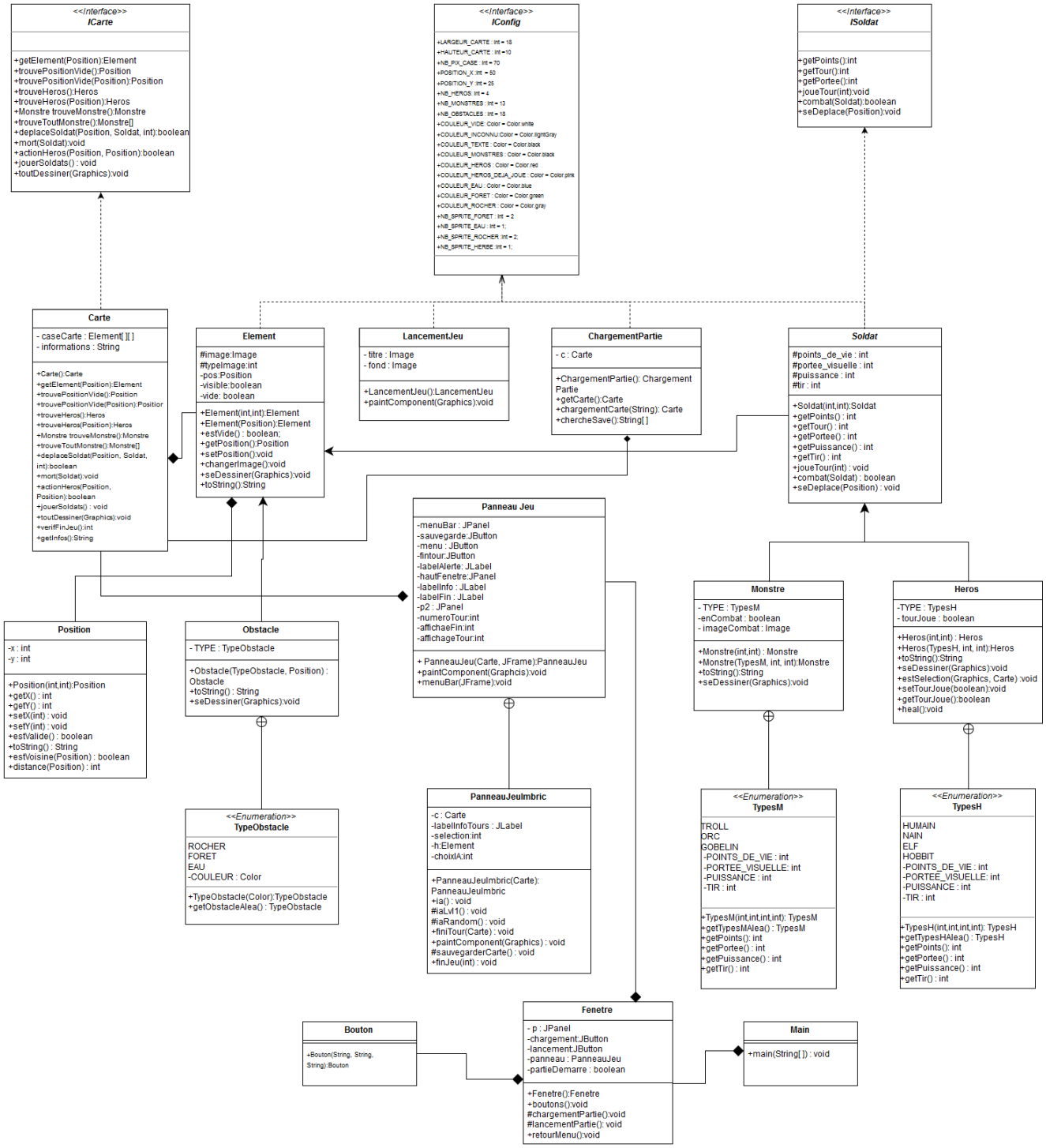
  Contient le détail de notre organisation du travail.

- Bilan

- Annexes

Analyse du projet

Diagramme de classes



(Disponible en png dans le dossier envoyé pour plus de lisibilité)

Explications

Nous avons ici tout d'abord repris la structure de base donnée (à savoir les 3 Interfaces IConfig, ISoldat et ICarte ainsi que les classes Element/Soldat etc..)

En interprétant l'énoncé du sujet et après concertation ainsi qu'avec diverses évolutions durant le développement de ce projet nous sommes arrivés à obtenir cette structure de classes.

Ainsi nous pouvons repérer 3 classes principales:

- Carte
- Element
- PanneauJeu

1- La classe Carte sera la pour gérer toutes les actions faites sur celle-ci, à savoir toute action du joueur pendant la partie (action, déplacer, attaquer etc).

Elle permettra également d'obtenir les différentes informations sur ses propres composants (Elements vide ? Chercher un Héros, un Monstre etc). Celle-ci bien évidemment se génèrera aléatoirement dans son constructeur à partir des valeurs de IConfig (nombre de monstres, d'obstacle etc).

2- La classe Element quant-à-elle est primordiale, la carte sera composés d'Element, ceux-ci pouvant être vide (Element même) ou bien un Soldat ou un Obstacle (héritage). Le Soldat lui-même pourra être un Héros (joueur) ou un Monstre (général-ordinateur) (devenant ainsi une classe abstraite). Ainsi un Element aura diverses valeurs, comme ça visibilité (brouillard de guerre) et le fait qu'il soit vide ou non.

Les soldats quant à eux auront les propriétés du type énumération, c'est-à-dire les caractéristiques de tout personnage de jeu (wargame) : points de vie, puissance au corps à corps, puissance de tir (distance), portée visuelle etc..

Chaque soldat dispose également d'un type (orc,elfe…) afin d'être reconnu et pour un affichage adapté.

Notre choix de structure a plus divergé quant à l'organisation des interfaces.

3- La classe PanneauJeu est la plus complexe et est également la classe qui contient tout le déroulement du jeu. C'est-à-dire qu'elle hérite de JPanel. C'est donc une interface, ou toute action de l'utilisateur (clic, mouvement de la souris) répercutera une action ou une autre (méthodes de Carte, Soldat…) suivant l'état du jeu actuel, représenté par ses différentes variables (Carte, tour de jeu…). Celle-ci gère également tout l'organisation et la structure des éléments graphique (swing) JPanel, JLabel, JButton grâce aux différentes possibilités de Layout associées.

C'est donc grâce à cette classe que l'utilisateur pourra interagir avec le jeu.

Pour autant celui-ci n'est pas atteint immédiatement, effectivement la classe Main lancera le jeu en créant une JFrame (Fenetre) correspondant à la page Menu (permettant de charger > classe associée, ou lancer une nouvelle partie). La classe Bouton ou encore LancementJeu sont des classes de configuration permettant une condensation du code et une lisibilité améliorée.

Techniques de POO mises en oeuvre

Héritage, encapsulation, polymorphisme, exceptions etc.

- **Exceptions**
  Nous avons utilisé des `try catch` pour tout ce qui est entrée/sortie de fichiers et pour le chargement des images.

- **Cascade**
  Certains constructeurs sont créés en cascades afin de permettre différents ajouts de paramètres (par exemple ceux de Héros). La redéfinition de méthodes comme toString a également été utile pour l'affichage, ainsi.

- **Héritage**
  L'héritage est principalement utilisé via la structure de base donnée de base :
  - Heros/Monstre qui héritent de Soldat (alors abstraite)
  - Soldat/Obstacle héritent de Element
  - De nombreuses classes implémentent les interfaces (IConfig, ISoldat, ICarte) contenant toutes les valeurs de paramétrage du jeu.
  - etc...

- **Polymorphisme**
  Notamment utilisé pour les interfaces et la classe imbriquée, n'existant pas au niveau d'au-dessus, un upcasting a lieu par la suite. De la même façon pour les Elements qui sont souvent cast dans leur classe par la suite : Nous connaissons l'existence d'un Element mais ne savons pas encore sa nature (actionHeros).

De manière générale nous avons essayé d'utiliser au maximum le système de programmation orientée objet qui est celui des classes, et de limiter à tout prix les variables publiques pour favoriser l'utilisation de méthodes adaptées (get/set ou autres).

Certains principes particuliers sont notamment ceux des classes imbriquées (utilisé pour une simplification d'utilisation dans PanneauJeu comme il l'était recommandé de le faire en TP).

Présentation des résultat

Fonctionnalités / Interface

- **Menu**
  Un menu permet de choisir de lancer une partie ou de charger (par clic ou raccourcis), celui-ci se compose donc d'un JPanel et de JButton et contient une image de fond. L'utilisation d'un GridBagLayout a pu ici être utile à la disposition des éléments.

  Il s'agit de la première interface en contact avec l'utilisateur lors du lancement du jeu, mais celle-ci peut être atteinte à tout moment pour arrêter une partie avec le bouton (ou raccourcis) associé une fois le jeu lancé.

- **Gestion des sauvegardes multiples**
  Quand on sauvegarde on crée dans le dossier `save` un nouveau fichier sauvegarde défini par la date.
  Quand on charge (géré dans `ChargementPartie.java`):
  - S'il n'y a pas de sauvegarde on fait une alerte
  - S'il y a des sauvegarde on extrait la liste des sauvegardes et on donne le choix avec un `JOptionPane`

- **Image des éléments**
  Les Elements de la carte ne sont plus représentés par des couleurs mais par des images, les obstacles quant-à-eux, ont une image choisie aléatoirement parmi un panel correspondant à leur type (eau,rocher,foret...). (Images créées)

  Une image indiquant le combat a également été implémentée, l'indication q'un Héros a joué n'est quant à lui que l'ajout d'un rectangle d'opacité modifiée par-dessus l'image.

- **Gestion de raccourcis claviers basiques**
  Quelques raccourcis ont été ajoutés au jeu :

  - On peut revenir en menu depuis la partie avec le bouton `Echap`
  - On peut également depuis le menu choisir de charger ou lancer une partie avec les touches 1 et 2 ou de quitter avec `Echap`.

- **Sélection des unités**
  On peut sélectionner une unité avec le clic gauche, et soit donner un ordre (action) avec le clic droit, soit changer de sélection en cliquant ailleurs.

  On affiche lors d'une sélection les cases visibles (portée d'attaque) de l'unité d'une couleur, et les cases de déplacements possibles d'une autre couleur. (Les Héros alliés, Monstres ennemis, et Obstacles sont également différenciés en étant soit ignorés dans l'affichage (obstacle) soit encadrés).

  Le Héros actuel en sélection est également mis en avant. Et celui ayant déjà joué sera grisé.

- **Label alerte**
  Un label alerte informe l'utilisateur (sous la barre de menu) tel que celui-ci affiche l'action effectuée par l'ennemi ou le héros (combat, déplacement) mais aussi une action incorrecte ou la fin d'un tour (numéro de tour).

  Ce label informe donc de l'action en cours (combat, mouvement) mais également du résultat de celle-ci (hors de portée, position obstacle etc.)

- **IA**
  Deux IA possible:
  - Une IA aléatoire comme demandée par le sujet (choixIA=1) : Si un ennemi est a portée, il attaque sinon se déplace aléatoirement sur les cases disponibles autour de lui.
  - Une IA améliorée (choixIA=2): Celle-ci repère tous les Héros à portée du groupe de Monstres et chaque Monstre essayera de rejoindre ou attaquer si possible l'ennemi le plus près de lui. Un Monstre peut également se régénérer si celui-ci est hors de portée d'attaque et n'a pas tout sa vie. En attente d'un des Monstre aie repéré un Héros, ceux-ci balayent la carte de gauche à droite.

- **Brouillard de Guerre et déplacement**
  Nous avons décidé de modifier la vue des Héros.
  Ainsi une portée de 3 ne représente non plus un carré de 7 par 7 centré sur le Héros mais un losange, représentant ainsi une vision plus proche d'une portée en cercle (plus réaliste), et de la même façon les Héros ne peuvent plus se déplacer que sur les 4 cases les entourant, et non plus en diagonale.
  Une exception a été faite pour le nain, de par sa contrainte d'avoir une portée visuelle de 1 (1 case), nous avons décidé de lui accorder la vision (l'attaque) ainsi que le déplacement sur les 8 cases l'entourant.

- **Interfaces générales**
  Nous avons décidé par raison de praticité d'imbriqué une classe dans une autre (dans PanneauJeu), afin de délimiter les actions et leurs impacts, notamment pour l'utilisation de boutons, et par clarification de la structure d'imbrication de JPanel-s et JLabel-s (et JButton-s). Cette classe imbriquée est celle ou se déroule toutes les actions concernant le jeu lui-même (les écouteurs d'évènement au clic, au mouvement de la souris etc.).

  Les autres Labels et JPanels sont dans les classe Fenetre, LancementJeu et Bouton.

Temps de travail et répartition

Christopher JEAMME
Temps: Environ 50h Travaillé majoritairement sur:

- Partie graphique
- Système de sauvegarde
- Ecouteurs
- Généralités

Romain PEURIERE
Temps: 40h+ Travaillé majoritairement sur:

- Gestion des interfaces et dispositions
- Intelligence artificielle
- Généralités

Dimitri BRUYERE
Temps: Environ 30 heures Travaillé majoritairement sur:

- Fonctions de jeu
- Organisation des classes
- UML

Notre but n'a pas été ici de délimiter dès le début 3 parts totalement séparée du travail mais au fur et à mesure durant le développement de ce projet de voir quelles sont les nécessités, le travail qui devait être accompli et à ce moment-là de choisir qui l'effectuera.

Ainsi nous avons très bien pu travailler tous sur les mêmes fonctions et les modifier chacun (notamment au début du projet et pendant le temps de mise en place de celui-ci), que travailler sur des fonctionnalités à part entière chacun de notre côté.

Bilan

Nous sommes parvenus à créer un jeu de wargame en java conformément aux prérequis de sujet. Celui-ci implémente donc les fonctionnalités de base mais également d'autres comme une IA améliorée, une interface graphique plus évoluée (un menu, des images, une sélection des Heros visible...) ainsi qu'une gestion de la sauvegarde permettant à l'utilisateur de sauvegarder et reprendre différentes parties. Les raccourcis clavier ont également été implémentés comme décrit plus haut etc.

De très nombreuses améliorations auraient été également possibles, là où nous avons préféré implémenter plus de fonctionnalités.

- D'un point de vue purement "code" nous aurions pu faire de nombreuses améliorations pour la rapidité. Notamment éviter les balayages de la carte (boucles for) et ainsi trouver une représentation (composition) alternative de la Carte (tableaux de Héros, Monstres, Obstacles) plus efficace mais moins intuitive aux premiers abords. Une IA plus perfectionnée etc.
- D'un point de vue POO, certains principes auraient pu être améliorées, ainsi faire passer la classe Element en abstraite tout en créant une classe ElementVide afin de condenser certaines parties de code. De la même façon pour la gestion des classes d'Interface (JPanel et ainsi de suite). Nous aurions également pu utiliser d'avantage les exceptions (try catch) et en créer des personnalisées de manière adaptée au jeu.
- D'un point de vue fonctionnalité, les possibilités sont infinies, mais des idées laissées en suspens ont été par exemple l'affichage d'informations sur le combat (point de vie perdus etc) ainsi que les fonctionnalités bonus proposées dans le sujet (sons, grilles hexagonales, plus de raccourcis clavier...).

D’autre part ce projet nous aura permis de mieux approfondir notre expérience et nos connaissances en Java, et de manière générale en Programmation Orientée Objet.

Annexes

Liste exhaustive des ressources utilisées

Outils, livres, tutoriels, site web

- Google.fr
- Images
  - Hobbit: Image basée sur http://www.otakia.com/wp-content/uploads/V_1/article_3723/8484.jpg
- https://openclassroom.com
- https://developpez.net/forums/
- https://docs.oracle.com