Licence d'Informatique 3ème année

Projet de POO : Wargame

1) Introduction

Le projet de POO consiste à programmer en Java un jeu de stratégie permettant de jouer à la guerre dans le monde (imaginaire) du seigneur des anneaux.

Les sections suivantes présentent le cœur de l'application puis l'interface graphique. Une organisation en classes et interfaces est **proposée**, ainsi que le fonctionnement de quelques méthodes, mais sans détailler chaque classe, chaque variable et chaque méthode. Vous aurez donc des choix de conception et d'implémentation à faire, pour compléter et éventuellement modifier et améliorer cette proposition.

L'utilisation d'un environnement de développement (*Eclipse*, *NetBeans*, etc.) est recommandée.

2) Carte, armées, obstacles, etc. (sans interface graphique)

On souhaite réaliser un paquetage `wargame` définissant des classes permettant à un général-joueur dirigeant une armée de `Heros` (humains, nains, elfes, hobbits, etc.) d'affronter une armée de `Monstres` (trolls, orcs, gobelins, etc.) dirigée par le général-ordinateur.

Les soldats sont disposés sur une carte représentée par une grille rectangulaire. Chaque case (position) de la grille ne peut contenir qu'un seul élément : un soldat ou un obstacle naturel infranchissable (eau, rocher ou forêt).

Une partie est organisée en *tours de jeu*. À chaque tour de jeu, le général-joueur peut ordonner à chacun de ses soldats de se déplacer dans une des 8 cases adjacentes (à condition qu'elle soit vide), d'attaquer un ennemi à sa portée… ou de se reposer. Quand le joueur a terminé son tour de jeu, les monstres agissent chacun à leur tour. Si un monstre repère un héros à sa portée, il l'attaque. Sinon, il se déplace dans une case choisie aléatoirement parmi les 8 cases adjacentes ne contenant aucun élément (ni soldat ni obstacle naturel).

Les tours de jeu s'enchaînent jusqu'à ce qu'une des deux armées soient complètement anéantie.

Le paquetage `wargame` pourra être organisé de la manière suivante :

- L'interface `IConfig` rassemble les principaux paramètres du jeu ;
- Les interfaces `ICarte` et `ISoldat` donnent les signatures des méthodes de `Carte` et `Soldat` ;
- La classe `Carte` gère la carte et des éléments qui y figurent ;
- La classe `Position` gère la position d'un élément sur la carte ;
- Les classes `Element`, `Soldat`, `Heros`, `Monstre` et `Obstacle` gèrent les soldats et les obstacles ;
- La classe `PanneauJeu` gère l'affichage de la carte.
- La classe `FenetreJeu` contient le `main` et affiche la carte ainsi que d'autres informations du jeu.

Les trois interfaces `IConfig`, `ICarte` et `ISoldat`, ainsi que les classes `Position` et `Obstacle` sont disponibles sur le cours en ligne « POO L3 ». Elles pourront être complétées et/ou modifiées.

2.1) Armées

À chaque soldat (`Heros` et `Monstres`) correspond un type de soldat, un nombre de point de vie, une portée visuelle, une puissance de frappe et une puissance de tir (cf. interface `ISoldat` et en particulier les types énumérés `TypesH` et `TypesM`). Certains types de soldats ne savent pas combattre à distance (arcs, arbalètes, lance-pierres, etc.) : leur puissance de tir est alors égale à zéro.

La portée visuelle représente la distance (en nombre de cases) à laquelle un soldat est capable de repérer amis, ennemis ou obstacles, et éventuellement d'attaquer un ennemi. Exemple : une portée visuelle de 3 correspond à 3 cases dans chaque direction, donc un carré de 7x7 centré sur le soldat. L'union de toutes les portées visuelles d'une armée détermine l'étendue de la carte visible par son général.

Les soldats de chaque armée sont numérotés. Le type et la position des soldats sont tirés aléatoirement au démarrage du jeu (les héros dans la partie gauche de la carte, les monstres dans la partie droite).

Un soldat peut perdre des points de vie lors d'un combat, et en regagner en se reposant, mais il ne peut pas dépasser son nombre de points de vie initial. Un soldat dont le nombre de points de vie est inférieur ou égal à zéro meurt et disparaît du jeu. Attention, un soldat ne peut effectuer qu'une seule action par tour de jeu !

2.2) Carte

La classe `Carte` gère la carte et les éléments qui y figurent (cf. interface `ICarte`). La création et le positionnement aléatoire des soldats et des obstacles s'effectuent dans son constructeur.

2.3) Actions des héros

Le déplacement d'un soldat fait intervenir la méthode `deplaceSoldat` de la classe `Carte` (qui déplace le soldat dans la carte), et la méthode `seDeplace` de la classe `Soldat` (qui met à jour l'objet `Soldat`).

La méthode `boolean actionHeros(Position pos, Position pos2)` de la classe `Carte` est appelée quand le joueur donne l'ordre au héros situé à la position `pos` d'agir à la position `pos2`. Le héros s'y déplace si la position est vide, ou combat le monstre qui s'y trouve éventuellement. La méthode retourne `false` s'il n'y a pas de héros à la position `pos`, si `pos2` n'est pas une position adjacente, s'il y a déjà un héros à la position `pos2`, ou si le héros a déjà joué son tour.

2.4) Actions des monstres et fin du tour

Une fois que le joueur a donné ses ordres à tout ou partie de son armée, la méthode `jouerSoldats()` est appelée pour terminer le tour de jeu. Les héros qui n'ont pas reçu d'ordre ce tour-là se reposent, et les monstres combattent ou se déplacent comme expliqué en début de section.

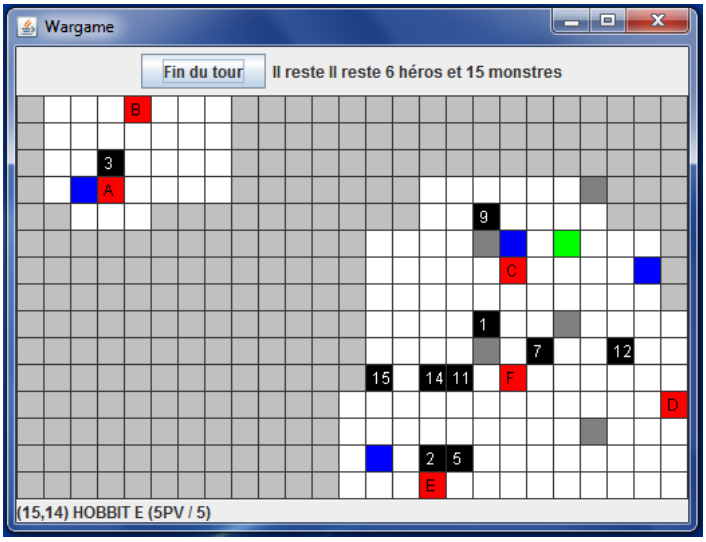
2.5) Combats

La méthode `combat(Soldat soldat)` de la classe `Soldat` implémente le déroulement d'un combat. Si les deux soldats sont dans deux cases adjacentes, il s'agit d'un corps-à-corps, et la puissance des coups portés est un tirage aléatoire dans l'intervalle [0..p] où p est la puissance de frappe des soldats. Sinon, il s'agit d'un combat à distance et p est la puissance de tir des soldats. Le soldat qui a lancé le combat porte la première attaque. Son adversaire perd un nombre de points de vie égal à la puissance de l'attaque. Ensuite, s'il est encore vivant, l'adversaire réplique.

La méthode `mort(Soldat soldat)` de la classe `Carte` est appelée en cas de décès brutal d'un combattant.

3) Interface graphique

On souhaite disposer de l'interface graphique basique ci-dessous (**au minimum !**).



Les soldats et les obstacles sont représentés sur la carte à l'aide d'un carré de la couleur correspondante (cf. `IConfig`), par exemple : eau = bleu, monstre = noir, héros = rouge (ou rose s'il a déjà joué ce tour-ci). Les monstres sont identifiés par leur numéro et les héros par la lettre correspondant à leur numéro. Les cases hors de portée visuelle de l'armée des héros sont grisées.

Le nombre de héros et le nombre de monstres encore en vie sont affichés au-dessus de la carte.

Quand le pointeur de la souris survole un élément, les détails le concernant sont

affichés sous la carte. Dans cet exemple, le pointeur de la souris survole le hobbit E situé dans la case (15,14) et disposant de 5 points de vie sur un maximum de 5.

Le général-joueur donne ses ordres aux héros (déplacement, attaque) par glisser-déposer. Le bouton « Fin du tour » permettra au général-joueur de signaler qu'il a terminé son tour de jeu.

Ne pas oublier de mettre à jour l'interface graphique à chaque changement !

4) Exceptions, sauvegarde et restauration

Utiliser le mécanisme des exceptions pour intercepter et traiter les principales situations exceptionnelles ou erreurs qui peuvent se produire dans l'application. Par exemple : tentative de déplacer un soldat sur une case occupée ou tentative de restauration d'une sauvegarde inexistante.

Ajouter à l'interface les fonctionnalités suivantes :

- Un bouton « redémarrer » permettant au joueur de redémarrer une nouvelle partie.
- Un bouton « sauvegarder » permettant de sauvegarder l'état du jeu dans un fichier `wargame.ser`.
- Un bouton « restaurer » pour reprendre la partie sauvegardée.
- Proposer un double accès aux fonctionnalités : via une barre de menu et via des boutons.

5) Améliorations

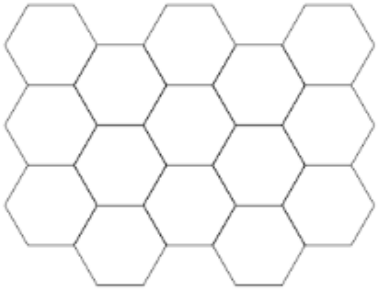
Vous ambitionnez une note significativement supérieure à 10 ? Améliorer l'application « de base ». Soyez imaginatifs : l'originalité et les astuces ergonomiques sont encouragées ! Vous pouvez vous inspirer de Wargames existants, et même réutiliser certaines ressources libres (icônes, images, sons, etc. Exemple : http://www.wesnoth.org/) à condition de le préciser dans votre rapport.

The Battle for Wesnoth (http://www.wesnoth.org/)

Par exemple, les améliorations suivantes seront valorisées :

- Proposer et implémenter une stratégie du général-ordinateur un peu plus efficace (et intéressante) que le comportement simpliste demandé dans la section 2.
- Afficher les détails concernant un soldat (en particulier son nombre de points de vie restant) dans une infobulle plutôt que sous la carte.
- Gérer correctement les obstacles : un soldat ne peut pas tirer une flèche à travers un arbre.
- Utiliser des images pour améliorer l'aspect visuel des boutons.
- Utiliser des images plutôt qu'un simple code-couleurs pour représenter les éléments du jeu.
- Utiliser des sons pour rendre le jeu plus vivant (musique, bruitages).
- Permettre de sauvegarder plusieurs parties différentes.

– Distinguer graphiquement les unités qui ont déjà réalisé une action ce tour-ci.
– Visualiser le rayon d'action et/ou de déplacement des unités.
– Permettre d'utilisation de raccourcis-clavier.
– Ajouter de nouveaux types de soldats (magiciens, soigneurs, catapultes, etc.).
– Ajouter de nouveaux types d'obstacles (montagne qui ralentit la progression, village qui améliore le repos, désert qui fait perdre des points de vie, etc.).
– Représenter la carte par une grille hexagonale.
– Permettre au joueur de personnaliser la carte (taille, nombre de soldats, d'obstacles, etc.).
– Proposer une génération aléatoire de la carte.



6) Rendu, rapport, soutenance

Vous devez déposer votre projet sur le cours en ligne « POO L3 » (espace « Travaux ») au plus tard le 11 décembre 2016 à 23h59 (sous peine de pénalités), sous la forme de 4 fichiers :

– Le rapport au format PDF.
– L'application complète sous la forme d'un fichier .jar exécutable (à tester !).
– Le code complet sous la forme d'un fichier .jar (incluant un fichier de sauvegarde et les éventuelles ressources graphiques utilisées).
– La documentation *Javadoc* sous la forme de fichiers HTML compressés dans un fichier zip.

Votre code doit :

– utiliser à bon escient les principes de la POO ;
– être compact, lisible, commenté ;
– avoir été soigneusement testé ;
– être prêt à compiler ;
– et enfin, pouvoir être compilé en ligne de commande sur l'environnement utilisé en TP, **sous peine de ne pas pouvoir être corrigé !**

Le rapport d'une dizaine de pages maximum (hors annexes) contiendra notamment :

– Une introduction qui décrit succinctement le but du projet et le plan du rapport,
– L'analyse du projet comprenant au moins un diagramme UML des classes commenté,
– Les techniques de POO/Java mises en œuvre (héritage, encapsulation, polymorphisme, exceptions, etc.),
– Une présentation synthétique du résultat de votre travail (fonctionnalités / interface),
– Une estimation du temps consacré au projet par chaque étudiant, ainsi que de l'organisation de votre travail (répartition des tâches au sein du trinôme, etc.),
– Une liste exhaustive des ressources utilisées (outils, livres, tutoriels, sites Web),
– Une conclusion comportant un bilan critique de votre travail (résultats, points forts / faibles, etc.), ainsi que vos impressions sur les aspects de la POO qui vous ont particulièrement intéressé (ou non), que vous avez trouvé plus difficile, etc.

Chaque trinôme doit réaliser son propre projet. Le plagiat de rapport ou de code (même partiel), ne sera pas tolérés : les plagieurs et les plagiés seront sanctionnés.

Les soutenances des projets auront lieu le 14/12 de 14h à 18h en salle A24.
Déroulement d'une soutenance : présentation avec diapos (Powerpoint ou autre, sur clé USB ou sur un portable), 10 minutes par groupe plus 5 minutes de questions. Le temps de parole doit être équitablement réparti entre les membres du groupe. Vous pouvez éventuellement préparer une petite démonstration, si vous venez avec votre portable et si ça présente vraiment un intérêt par rapport à de "simples" copies d'écran.
L'objectif de la soutenance est de mettre en valeur votre travail, sans forcément reprendre tous les éléments de votre rapport, en présentant entre autres :

– Fonctionnalités que vous avez implémentées, bilan ergonomique ;
– Technique : diagramme des classes, structure du code, techniques POO ;
– Organisation du travail (temps, groupe, partage des tâches) ;
– Problèmes rencontrés, limitations ;
– Conclusion, bilan critique de votre travail.