Rapport de Projet

Programmation Orientée Objet

Licence 3 Informatique

TURNEL Mickaël

LEQUIN Jonathan

POUVARET Line

2013-2014

Introduction

Le but du projet était de réaliser un jeu de stratégie en Java permettant de jouer à la guerre dans le monde imaginaire du Seigneur des Anneaux. Le « général » joueur dispose d'une armée d'humains, d'elfes, de nains et de hobbits tandis que le « général » ordinateur dispose d'une armée d'orcs, de trolls et de gobelins. Le but du jeu est de tuer tous les monstres.

Le jeu devait être réalisé du 4 novembre 2013 au 8 décembre 2013 à 23h59 et en trinôme.

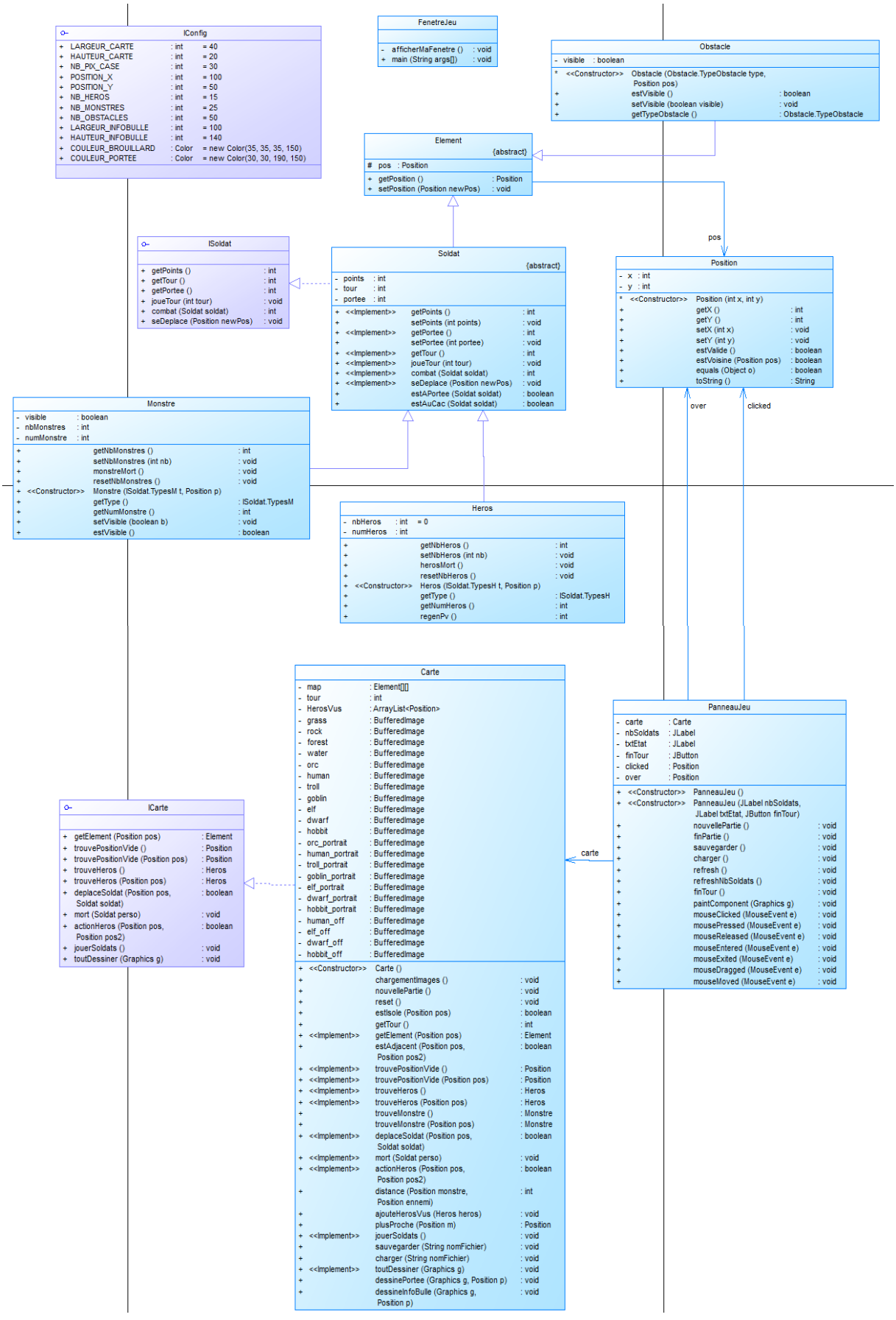
Plan

Introduction

I. Analyse
II. Techniques
III. Synthèse
IV. Organisation
V. Ressources

Conclusion

I – Analyse


IConfig
+ LARGEUR_CARTE : int = 40
+ HAUTEUR_CARTE : int = 20
+ NB_PIX_CASE : int = 30
+ POSITION_X : int = 100
+ POSITION_Y : int = 50
+ NB_HEROS : int = 15
+ NB_MONSTRES : int = 25
+ NB_OBSTACLES : int = 50
+ LARGEUR_INFOBULLE : int = 100
+ HAUTEUR_INFOBULLE : int = 140
+ COULEUR_BROUILLARD : Color = new Color(35, 35, 35, 150)
+ COULEUR_PORTEE : Color = new Color(30, 30, 190, 150)
FenetreJeu
- afficherMaFenetre () : void
+ main (String args[]) : void
Obstacle
- visible : boolean
* <<Constructor>> Obstacle (Obstacle.TypeObstacle type, Position pos)
+ estVisible () : boolean
+ setVisible (boolean visible) : void
+ getTypeObstacle () : Obstacle.TypeObstacle
Element
{abstract}
pos : Position
+ getPosition () : Position
+ setPosition (Position newPos) : void
pos
ISoldat
+ getPoints () : int
+ getTour () : int
+ getPortee () : int
+ joueTour (int tour) : void
+ combat (Soldat soldat) : int
+ seDeplace (Position newPos) : void
Soldat
{abstract}
- points : int
- tour : int
- portee : int
+ <<Implement>> getPoints () : int
+ setPoints (int points) : void
+ <<Implement>> getPortee () : int
+ setPortee (int portee) : void
+ <<Implement>> getTour () : int
+ <<Implement>> joueTour (int tour) : void
+ <<Implement>> combat (Soldat soldat) : int
+ <<Implement>> seDeplace (Position newPos) : void
+ estAPortee (Soldat soldat) : boolean
+ estAuCac (Soldat soldat) : boolean
Position
- x : int
- y : int
* <<Constructor>> Position (int x, int y)
+ getX () : int
+ getY () : int
+ setX (int x) : void
+ setY (int y) : void
+ estValide () : boolean
+ estVoisine (Position pos) : boolean
+ equals (Object o) : boolean
+ toString () : String
over
clicked
Monstre
- visible : boolean
- nbMonstres : int
- numMonstre : int
+ getNbMonstres () : int
+ setNbMonstres (int nb) : void
+ monstreMort () : void
+ resetNbMonstres () : void
+ <<Constructor>> Monstre (ISoldat.TypesM t, Position p)
+ getType () : ISoldat.TypesM
+ getNumMonstre () : int
+ setVisible (boolean b) : void
+ estVisible () : boolean
Heros
- nbHeros : int = 0
- numHeros : int
+ getNbHeros () : int
+ setNbHeros (int nb) : void
+ herosMort () : void
+ resetNbHeros () : void
+ <<Constructor>> Heros (ISoldat.TypesH t, Position p)
+ getType () : ISoldat.TypesH
+ getNumHeros () : int
+ regenPv () : int
Carte
- map : Element[][]
- tour : int
- HerosVus : ArrayList<Position>
- grass : BufferedImage
- rock : BufferedImage
- forest : BufferedImage
- water : BufferedImage
- orc : BufferedImage
- human : BufferedImage
- troll : BufferedImage
- goblin : BufferedImage
- elf : BufferedImage
- dwarf : BufferedImage
- hobbit : BufferedImage
- orc_portrait : BufferedImage
- human_portrait : BufferedImage
- troll_portrait : BufferedImage
- goblin_portrait : BufferedImage
- elf_portrait : BufferedImage
- dwarf_portrait : BufferedImage
- hobbit_portrait : BufferedImage
- human_off : BufferedImage
- elf_off : BufferedImage
- dwarf_off : BufferedImage
- hobbit_off : BufferedImage
+ <<Constructor>> Carte ()
+ chargementImages () : void
+ nouvellePartie () : void
+ reset () : void
+ estIsole (Position pos) : boolean
+ getTour () : int
+ <<Implement>> getElement (Position pos) : Element
+ estAdjacent (Position pos, Position pos2) : boolean
+ <<Implement>> trouvePositionVide () : Position
+ <<Implement>> trouvePositionVide (Position pos) : Position
+ <<Implement>> trouveHeros () : Heros
+ <<Implement>> trouveHeros (Position pos) : Heros
+ trouveMonstre () : Monstre
+ trouveMonstre (Position pos) : Monstre
+ <<Implement>> deplaceSoldat (Position pos, Soldat soldat) : boolean
+ <<Implement>> mort (Soldat perso) : void
+ <<Implement>> actionHeros (Position pos, Position pos2) : boolean
+ distance (Position monstre, Position ennemi) : int
+ ajouteHerosVus (Heros heros) : void
+ plusProche (Position m) : Position
+ <<Implement>> jouerSoldats () : void
+ sauvegarder (String nomFichier) : void
+ charger (String nomFichier) : void
+ <<Implement>> toutDessiner (Graphics g) : void
+ dessinePortee (Graphics g, Position p) : void
+ dessineInfoBulle (Graphics g, Position p) : void
carte
PanneauJeu
- carte : Carte
- nbSoldats : JLabel
- txtEtat : JLabel
- finTour : JButton
- clicked : Position
- over : Position
+ <<Constructor>> PanneauJeu ()
+ <<Constructor>> PanneauJeu (JLabel nbSoldats, JLabel txtEtat, JButton finTour)
+ nouvellePartie () : void
+ finPartie () : void
+ sauvegarder () : void
+ charger () : void
+ refresh () : void
+ refreshNbSoldats () : void
+ finTour () : void
+ paintComponent (Graphics g) : void
+ mouseClicked (MouseEvent e) : void
+ mousePressed (MouseEvent e) : void
+ mouseReleased (MouseEvent e) : void
+ mouseEntered (MouseEvent e) : void
+ mouseExited (MouseEvent e) : void
+ mouseDragged (MouseEvent e) : void
+ mouseMoved (MouseEvent e) : void
ICarte
+ getElement (Position pos) : Element
+ trouvePositionVide () : Position
+ trouvePositionVide (Position pos) : Position
+ trouveHeros () : Heros
+ trouveHeros (Position pos) : Heros
+ deplaceSoldat (Position pos, Soldat soldat) : boolean
+ mort (Soldat perso) : void
+ actionHeros (Position pos, Position pos2) : boolean
+ jouerSoldats () : void
+ toutDessiner (Graphics g) : void

II – Techniques

Les techniques de Java mises en œuvre ici sont :

- L'héritage : exemple Soldat hérite de Element.
- La composition : exemple Element est composé d'un objet Position.
- L'implémentation d'interface : exemple Soldat implément ISoldat.
- Les classes abstraites : exemple Soldat et Element sont abstrait, on n'a pas besoin de les instancier.
- Le polymorphisme : exemple la Carte possède un tableau à deux dimensions d'Element nous permettant de stocker soit un obstacle, soit un héros soit un monstre.
- Les exceptions : exemple à la sauvegarde ou au chargement d'un fichier, une exception est générée si le fichier n'est pas conforme. Le programme va donc réceptionner et afficher l'erreur indiquant le problème au lieu de l'afficher dans la console.

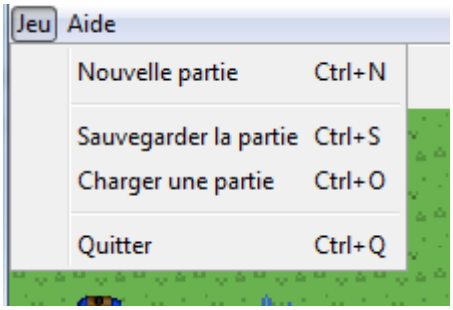
III – Synthèse

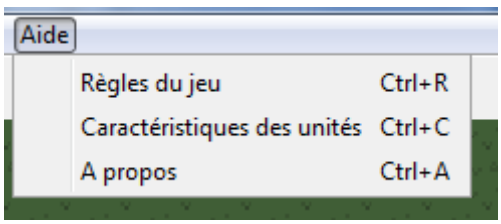
L'interface graphique

La fenêtre principale (1) est constituée de 4 parties.

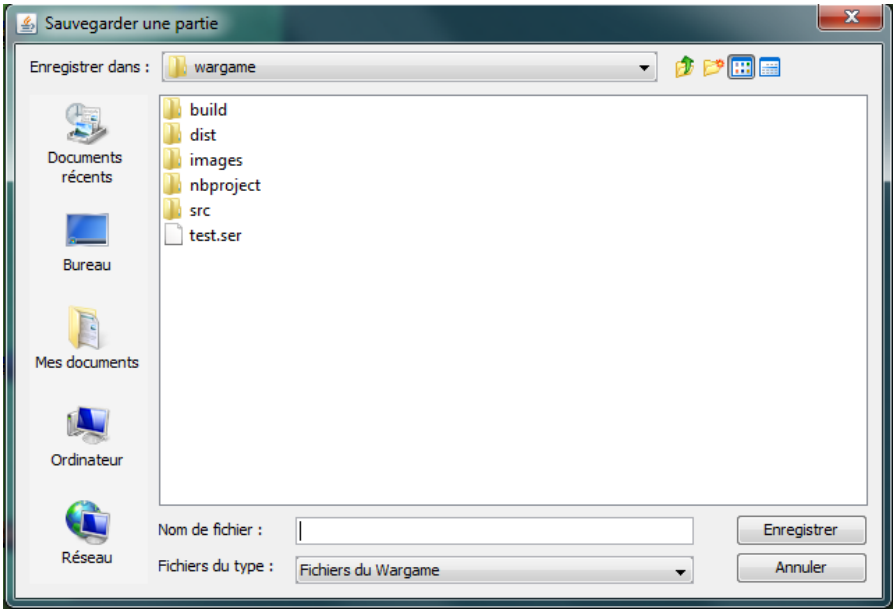
1. Une barre de menus (2)
2. Le bouton de fin de tour et un texte qui sert à afficher le nombre de héros et de soldats restant (3)
3. La carte du jeu (4)
4. Le texte qui sert à indiquer la position actuelle de la souris sur la carte (5)

1. La barre de menus

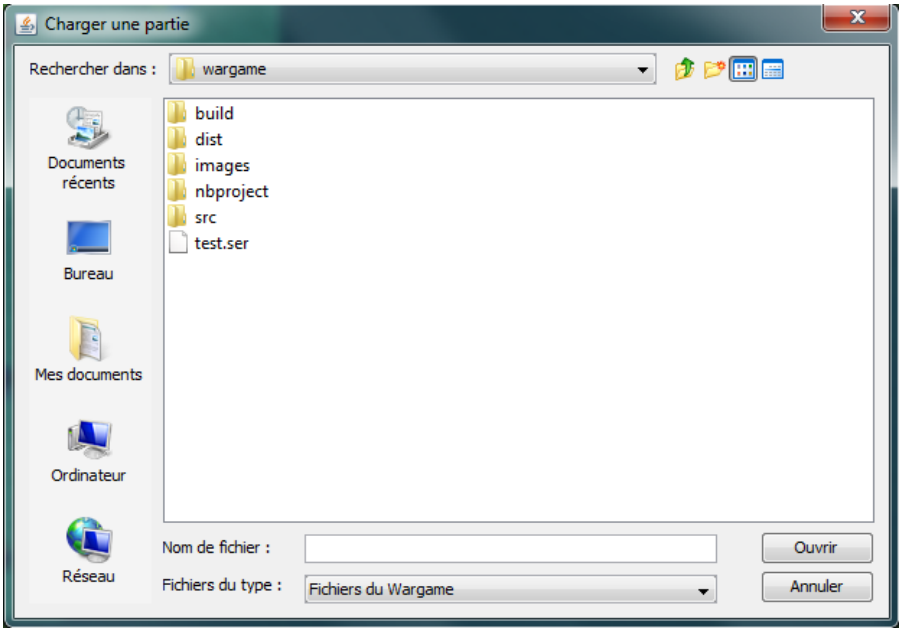




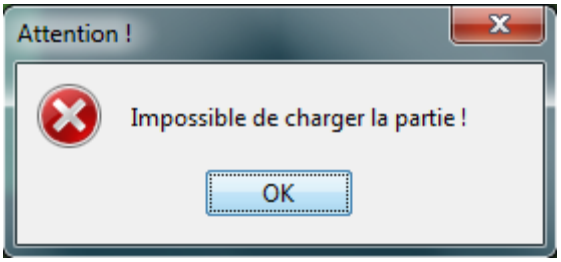
- Nouvelle partie : La carte va être mise à zéro et tout va être réinitialisé, les héros seront placés sur le tiers gauche de la carte, les monstres eux seront placés sur le tiers droit de la carte.
- Sauvegarder la partie : Une fenêtre va s'ouvrir pour proposer à l'utilisateur à quel endroit il veut sauvegarder sa partie. Les fichiers sauvegardés ont tous l'extension « .ser ». L'utilisateur ne peut voir seulement que les fichiers qui ont l'extension « .ser ».



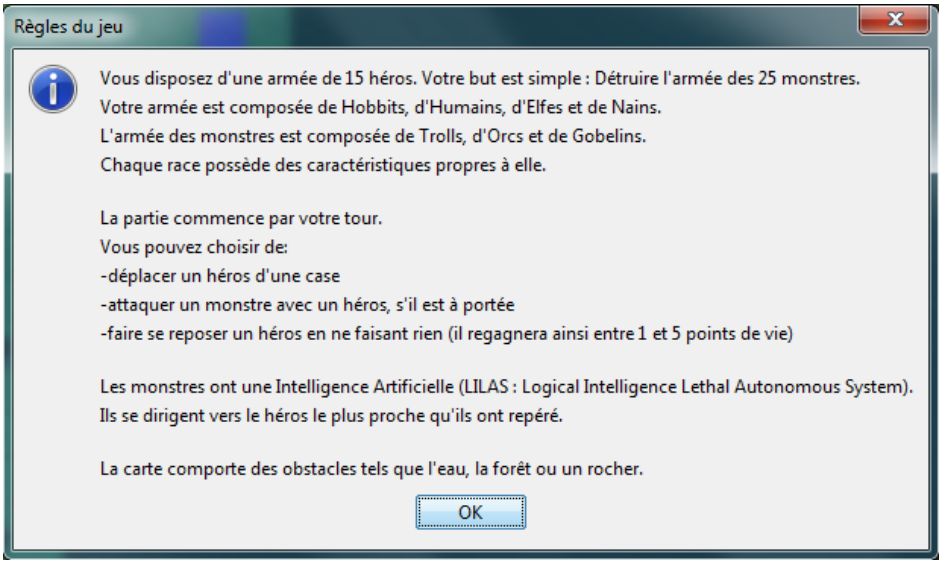
- Charger la partie : Une fenêtre va s'ouvrir pour proposer à l'utilisateur quel fichier il veut restaurer. De même que pour sauvegarder il ne pourra voir que les fichiers avec l'extension « .ser ».



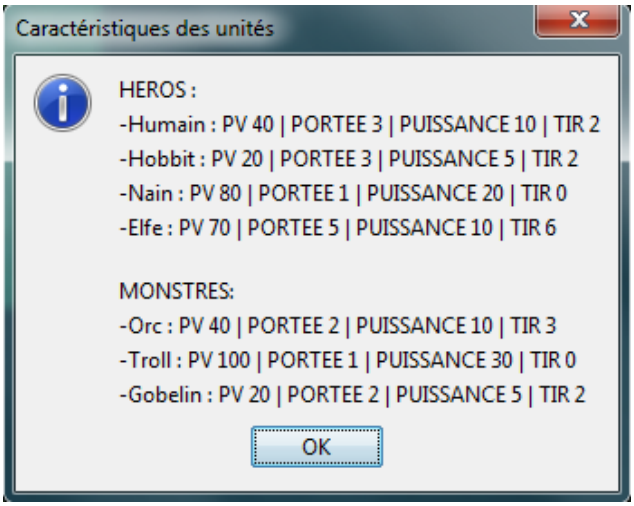
Si le chargement n'a pas pu se faire dû à une erreur dans le fichier de sauvegarde, une erreur apparaît.



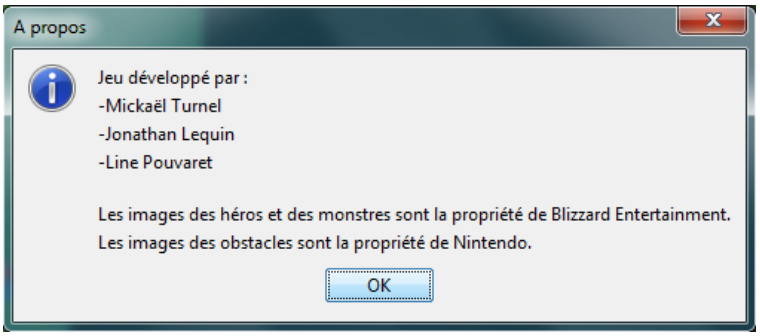
- Quitter : Ferme l'application.
- Règles du jeu : Une fenêtre va s'ouvrir avec l'explication du fonctionnement du jeu pour permettre à l'utilisateur de mieux s'y retrouver.



- Caractéristiques des unités : Une fenêtre va s'ouvrir pour montrer les caractéristiques de chaque unité du jeu.



- A propos : Une fenêtre va s'ouvrir pour présenter les auteurs du projet.



2. Fin de tour et affichage du nombre de soldats restants

Lors du clic sur le bouton « fin de tour », le tour actuel va prendre fin (cf. partie fonctionnement du jeu).

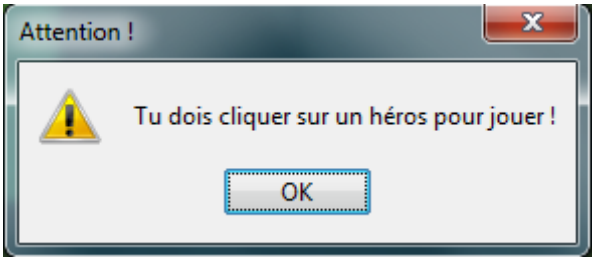
L'affichage du nombre de soldats restants va servir à indiquer au joueur combien il reste de héros en sa possession et de monstres restants sur la carte même s'il ne les voit pas.

3. La carte

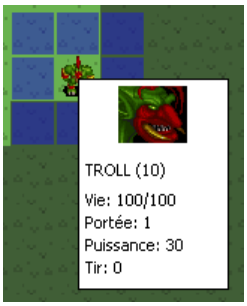
C'est le cœur du jeu, elle va afficher à l'utilisateur tout ce qui est en train de se passer, c'est-à-dire les déplacements, les informations des héros ou des monstres, leur portée, etc… .

Et c'est ici que l'utilisateur va pouvoir interagir avec le jeu.

Si l'utilisateur ne clique pas sur un héros, un message d'erreur va alors apparaître.

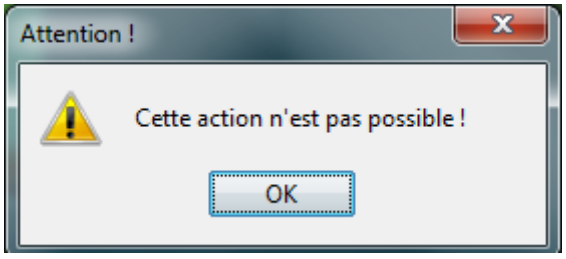


Si l'utilisateur passe sa souris au-dessus d'un héros, une info-bulle va s'afficher pour lui indiquer la vie actuelle du héros et ses informations. Le même principe s'applique à un monstre à la différence que sa portée va s'afficher en plus lors du passage de la souris.



Si l'utilisateur clique sur un héros la portée d'attaque de celui-ci va s'afficher. Il a alors deux choix, soit il clique sur une case adjacente à son héros pour un déplacement d'une case, soit il clique sur un monstre à portée.

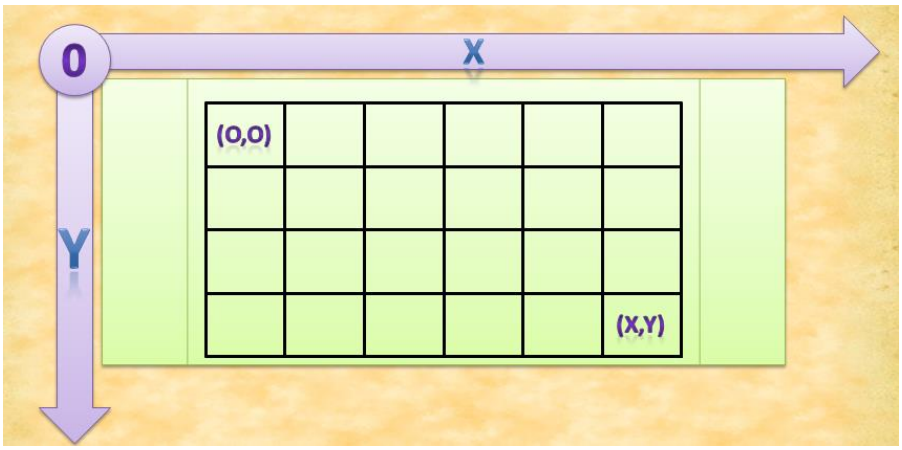
Si ce n'est pas le cas, alors un message d'erreur s'affiche.



4. La position actuelle de la souris

La carte fonctionne comme une grille et chaque composant sur la carte est une case.

La case (0, 0) est la case en haut à gauche.



Le texte en bas de la fenêtre va donc afficher la position actuelle de la souris sur la carte et indiquer à l'utilisateur sur quelle case se trouve sa souris.

Fonctionnement du jeu

Le but du jeu est que le joueur tue tous les monstres en utilisant les héros à sa disposition.

Lorsqu'un héros est joué, il doit attendre la fin du tour pour pouvoir jouer à nouveau.

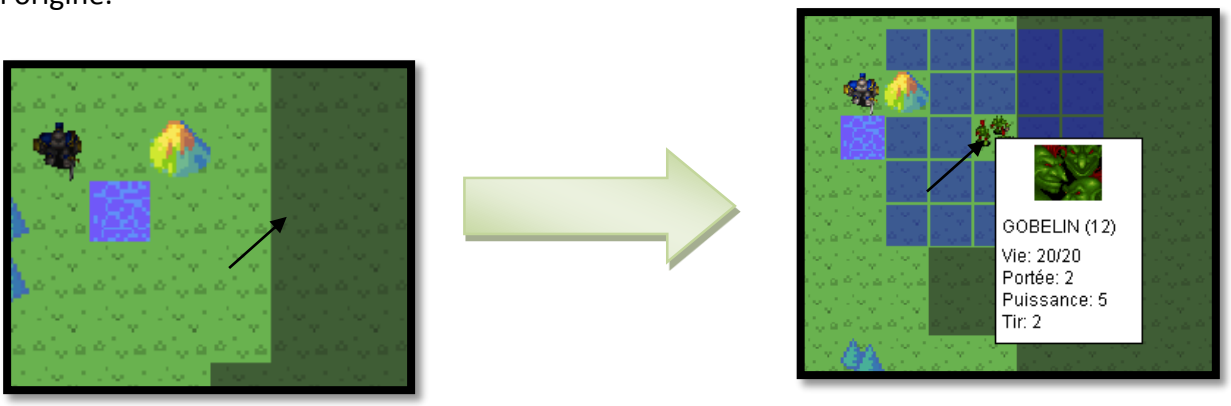
Le tour se termine lorsque l'utilisateur clique sur le bouton « fin de tour », c'est alors que les monstres font ce qu'ils ont à faire (cf. l'IA des monstres).

Cette partie va être décomposée en 3 sous parties :

1. Les héros / monstres
2. Le système de combat
3. L'IA des monstres

1. Héros / Monstres

Chaque monstre ou héros est défini par un type, quatre pour les héros (Humain, Elfe, Hobbit, Nain) et trois pour les monstres (Troll, Gobelins, Orc). En fonction de ce type, les Monstres et Héros vont se voir attribuer des caractéristiques (Points de vie, portée, puissance d'attaque, puissance de tir). De plus ceux-ci possèdent également un numéro unique découlant naturellement du nombre de monstres ou héros présents à l'origine.



Les monstres disposent également d'un attribut concernant leur visibilité servant à afficher ou masquer leurs info-bulles lorsque ceux-ci sont dissimulés par le brouillard de guerre. Il apparaissait en effet gênant de pouvoir détecter et éviter ou encercler des monstres supposément invisibles dont la présence aurait été trahie par l'info-bulle apparaissant lorsque le curseur serait passé sur eux.

Il était inutile d'implémenter cet attribut sur les héros, le jeu n'étant pas « réversible » il n'est pas possible de jouer les monstres contre les héros ceux-ci sont donc toujours visibles et le jeu ne gère donc pas leur visibilité.

Les héros disposent cependant de la capacité de régénérer leurs points de vie lorsqu'ils n'effectuent aucune action dans le tour. Ils peuvent dès lors régénérer entre 1 et 5 points de vie aléatoirement (pour un total n'excédant pas le maximum de points de vie).

2. Le système de combat

Le système de combat est assez simple, lorsque l'utilisateur clique sur un héros il aperçoit la portée du héros et si l'utilisateur clique ensuite sur un monstre le combat est engagé.

Voici le fonctionnement du combat :

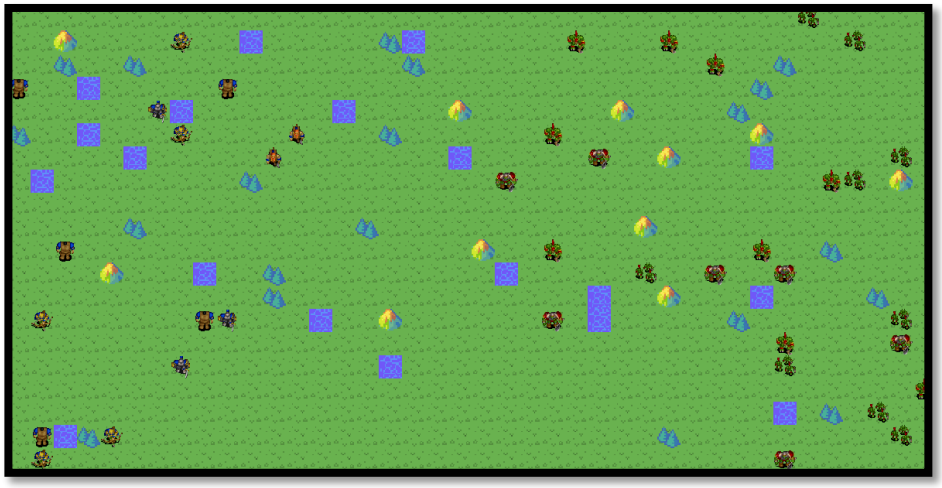
On teste d'abord si les deux soldats sont au corps à corps ou à distance.

- S'ils sont au corps à corps, la puissance d'attaque va servir à définir la puissance du coup qui est entre 1 et la puissance d'attaque. Le premier va donc porter un coup au second et le second va répliquer automatiquement car il est obligatoirement à portée.
- S'ils sont à distance, la puissance de tir va servir à définir la puissance du tir qui est entre 1 et la puissance de tir. Le premier va donc porter un tir au second, car la méthode est appelée seulement si le premier est à portée du second, mais le second ne répliquera que s'il est à portée du premier, s'il ne l'est pas il ne répliquera donc pas.

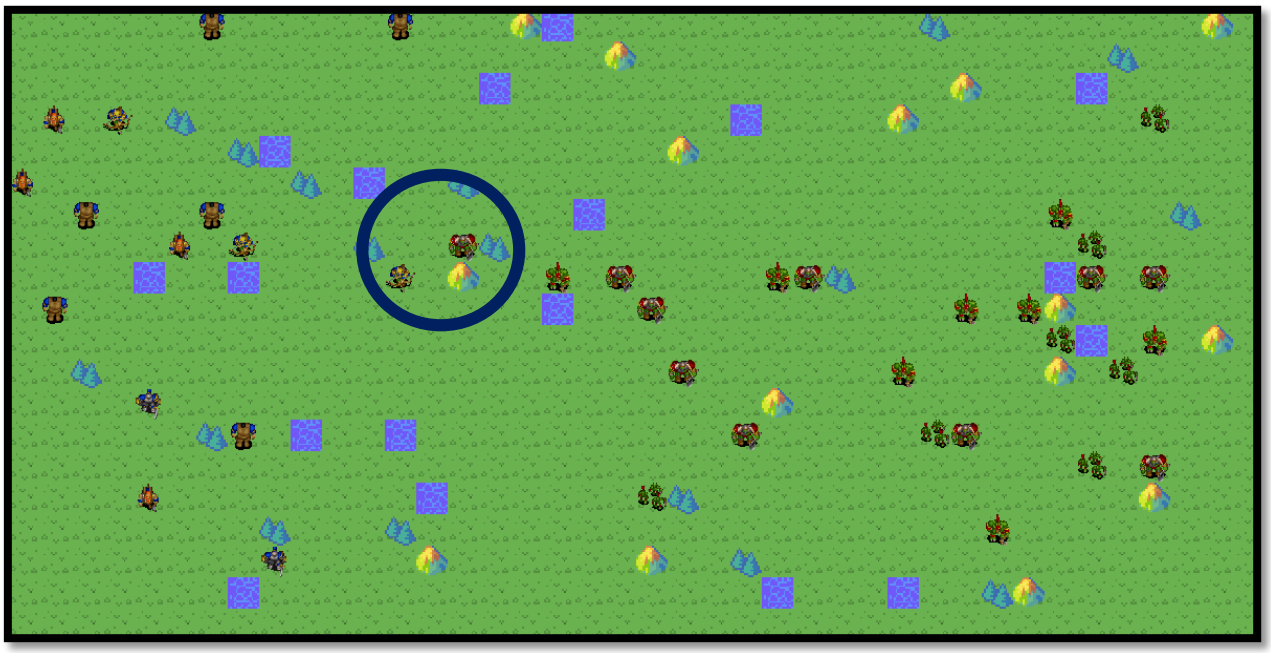
Lors de chaque coup donné, il y a une faible chance que le premier ou le second rate son coup. En effet lors de chaque coup, les soldats ont une probabilité de rater leurs coups de 5%. Il est plus logique de rater son coup avec un petit pourcentage plutôt que de donner un coup entre 0 et la puissance car avec cette méthode le soldat à autant de chance de n'infliger aucun dégât que d'infliger le maximum de dégâts.

3. L'IA des monstres : LILAS (Logical and Intelligent Lethal Autonomous System)

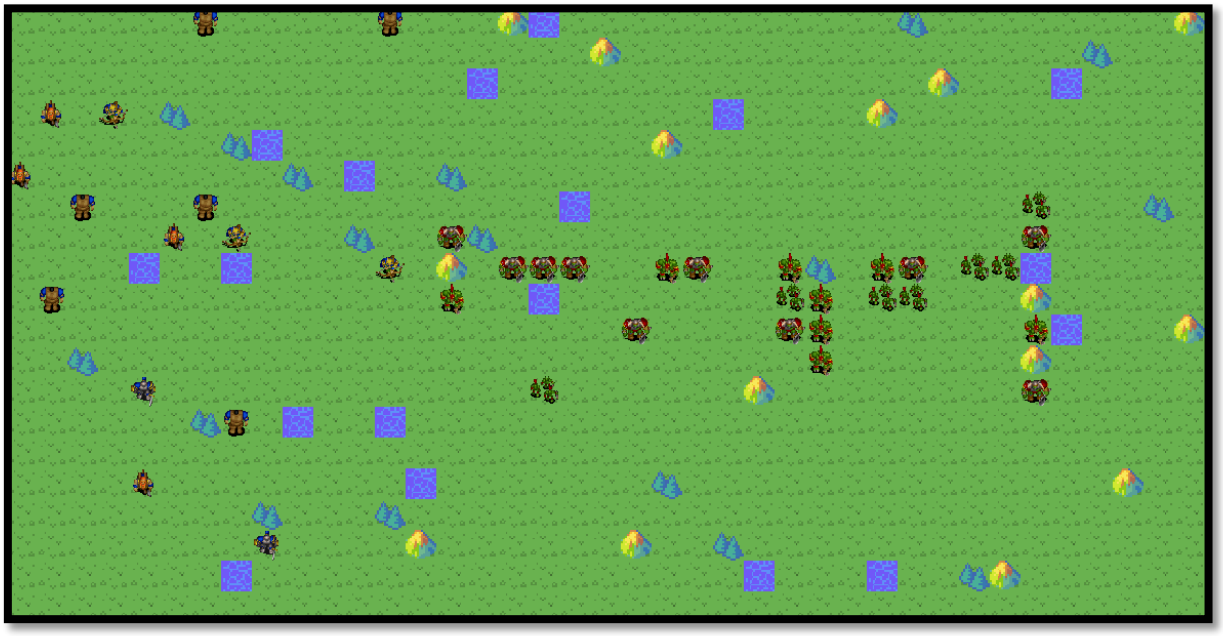
Nous avons également implémenté une IA simple mais plus développée que le déplacement aléatoire demandé.



Nous avons ajouté une liste de héros repérés par les monstres, celle-ci est évidemment vide lorsque le jeu démarre. Mais dès lors qu'un monstre attaque un héros qui est passé à sa portée, la position de ce dernier est ajoutée à la liste. Les monstres n'ayant pas de héros a portée vont alors se diriger vers le héros repéré le plus proche d'eux.



Ceci permet de donner une dimension beaucoup plus stratégique au jeu, car il ne suffit plus de simplement tuer un monstre qui serait passé trop près, mais également de tuer sans être repéré ou de se recacher pour ne pas se retrouver submerger par une armée de monstre qui deviennent vite ingérables en trop grand nombre.



Cela encourage aussi le joueur à déplacer ses pions afin de se disperser pour ne pas avoir tous ses pions découverts en une fois.

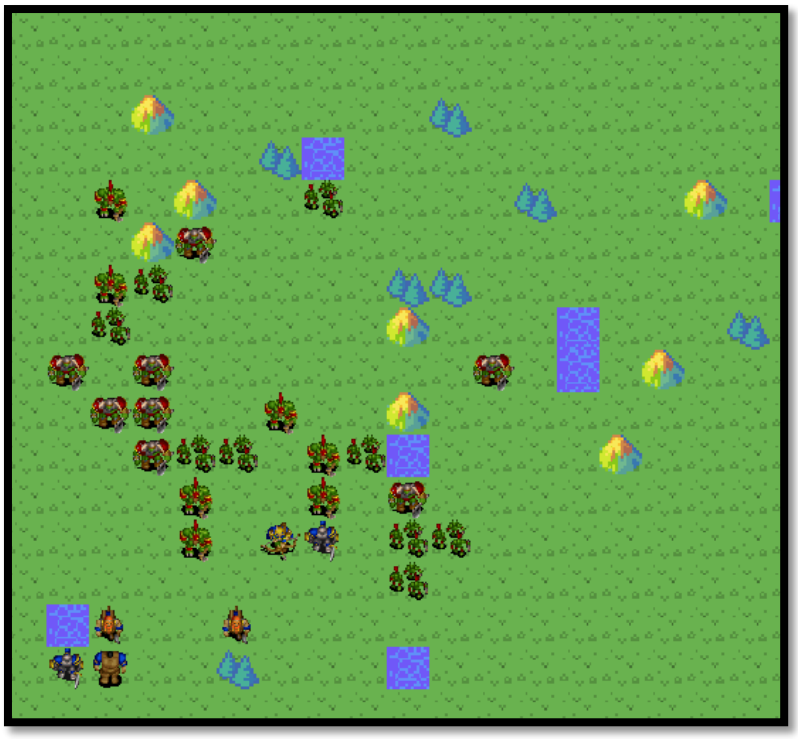
L'IA gère de façon assez rudimentaire les obstacles, si le héros repéré le plus proche se trouve à sa gauche elle pourra se déplacer vers la case immédiatement à gauche ou les cases respectivement en haut à gauche et en bas à gauche. Elle ne se retrouve donc bloquée que lorsqu'un mur d'obstacles se trouve devant-elle.

Il était cependant nécessaire de gérer la gêne entre monstres. En effet lorsque le premier héros est repéré, toute l'armée se dirige sur lui. Il est donc assez fréquent que les monstres se bloquent entre eux, les monstres ne pouvant se déplacer restent alors immobiles durant le tour.

Si pour une raison (déplacement par le joueur ou mort) un héros n'est plus présent sur la case de la liste, il est supprimé de celle-ci.

Le joueur peut donc utiliser ceci pour déplacer le héros repéré et ainsi se "camoufler" mais il perd un tour de jeu pour ce héros.

Si plus aucun héros n'est repéré l'IA passe alors à nouveau en déplacement aléatoire jusqu'à trouver le prochain héros.



IV – Organisation

Dès le début du projet, nous avons assigné chacun d'entre nous à l'implémentation de certaines classes et méthodes.

Au tout départ, Jonathan s'est chargé d'implémenter les classes **Heros** et **Monstre**, Mickaël la classe Soldat et la classe abstraite **Element** et Line la classe **Carte** qui promettait d'être plutôt longue.

Même si Line a implémenté la majeure partie de la classe **Carte**, celle-ci a été beaucoup retouchée par nous tous pendant tout le projet. Jonathan a ajouté des tests dans estAdjacent et a implémenté toute l'intelligence artificielle LILAS dans la méthode jouerSoldats. Mickaël a implémenté une grande partie de la méthode toutDessiner car elle touchait au composant graphique et il s'est occupé de la majeure partie de l'interface graphique. Les méthodes dessinePortee et dessineInfoBulle ont été entièrement réalisées par Mickaël car elles touchaient encore au composant graphique.

Line s'est chargée de trouver les images des héros, monstres et des obstacles et donc de rajouter leur affichage dans la classe **Carte**. Elle s'est également chargée d'en modifier certaines pour les griser (quand un héros a déjà joué, ou pour l'image du brouillard de guerre).

Mickaël a entièrement implémenté la classe **PanneauJeu** en nous expliquant pas-à-pas ce qu'il faisait à chaque fois (selon ce qu'on avait décidé ensemble pour notre interface graphique), pour nous permettre de la modifier si le besoin s'en ressentait. Line s'est chargée de l'implémentation des fonctions sauvegarder et charger puisqu'elles étaient reliées aux fonctions sauvegarder et charger de la classe **Carte**.

FenetreJeu a été surtout implémenté par Mickaël avec l'aide de Line pour l'implémentation de la rubrique Aide et le changement d'apparence de l'application en fonction du système d'exploitation utilisé.

Enfin, Jonathan a également entièrement réalisé la présentation PowerPoint et nous avons tous participé pour la rédaction du présent rapport. Mickaël s'est chargé d'ajouter la Javadoc dans le code source.

Pour ce qui est du temps passé par nous tous sur le projet, nous ne sommes pas vraiment arrivés à nous en rendre compte. Sûrement plus de 6 heures par semaine en moyenne ont été passées sur le projet pour chacun d'entre nous. Cela dépendait souvent de l'avancement des projets des autres matières mais étant donné que le sujet était motivant, nous nous mettions à travailler de bon cœur.

V – Ressources

Nous avons préféré utiliser NetBeans comme IDE plutôt qu'Eclipse car nous le trouvions plus intuitif, plus facile d'utilisation et plus rapide à lancer.

Pour pouvoir mettre en commun notre projet et travailler tous ensemble dessus à distance, nous avons utilisé un site internet : https://www.assembla.com sur lequel nous pouvions partager tous les fichiers du projet et les mettre à jour. Dès que l'un de nous faisait une modification sur un fichier, il suffisait de l'envoyer en indiquant éventuellement quelle modification avait été effectuée et nous pouvions donc tous récupérer le fichier mis à jour. Au total, il y a eu 132 révisions sur le site Assembla, c'est-à-dire 132 modifications du projet.

Pour pouvoir utiliser les images dans le jeu, nous avons eu besoin de l'aide de http://docs.oracle.com/ pour comprendre comment les charger puis les afficher à un endroit précis.

Finalement, le cours fourni par M. Gery nous a été d'une aide précieuse ainsi que l'expérience de Mickaël en Java grâce à la formation qu'il a reçue pour obtenir son DUT Informatique et durant laquelle il a été amené à manipuler diverses techniques en Java (par exemple les entrées et sorties, les sauvegardes d'objet dans un fichier, …).

Conclusion

En bilan, je pense qu'on peut tous les trois affirmer être assez fiers du travail que l'on a réalisé en un mois pour ce projet. Même si certaines idées et améliorations n'ont pas pu être rajoutées au projet par manque de temps, nous envisageons de faire évoluer le projet même après le rendu car cela a été pour nous un sujet très motivant.

Voici les idées que nous n'avons pas pu mettre en place :

- Le fait de pouvoir **choisir son équipe** (Héros ou Monstre) comme par exemple dans la plupart des jeux de rôle où on peut choisir entre deux factions (Alliance / Horde, République/Empire, etc..)
- Le fait de pouvoir jouer à **deux joueurs** en tour à tour (en masquant l'équipe adverse à chaque tour) ou bien en coopération contre l'ordinateur.
- Rajouter des **codes de triche** dans une sorte de console en bas de la carte (codes pour tuer les ennemis d'un coup, révéler la carte, gagner d'un coup, etc…)
- Rajouter une race du côté des monstres (exemple : mort-vivants, nazguls, etc..) ou bien modifier une race de sorte à ce qu'elle puisse soigner les personnages de sa faction (par exemple en cliquant sur un héros qui peut soigner, on clique sur un autre héros et cela le soigne).
- Rajouter des **dégâts de zone** pour certaines unités (touchant plusieurs cases).
- Rajouter des **options** en permettant à l'utilisateur de changer la taille de la carte, le nombre de héros, le nombre de monstres, la puissance des monstres, des héros, de désactiver l'IA LILAS, etc…

Notre jeu manque de certaines fonctions qui représentent ses points faibles :

- L'utilisation de **sons**, notamment au moment des combats (mort d'un héros/monstre, attaque ratée, attaque réussie, bruit d'une flèche pour les attaques à distance, etc…)
- Les **messages d'avertissement** où l'on doit cliquer sur « Ok » à chaque fois qu'on sélectionne un héros et qu'on veut cliquer sur une case incorrecte ensuite (non adjacente ou pas un monstre) ou bien quand on clique dans le vide sans avoir sélectionné de héros. Des gens extérieurs à la Faculté ont testé le jeu et ont avoué que ces messages étaient plutôt ennuyants.
- La **gestion d'obstacles** au moment des attaques. En effet, actuellement, les obstacles ne servent qu'à bloquer des déplacements, ce qui est un peu dommage à notre sens. Nous voulions rajouter le fait qu'un héros (ou un monstre) ne pourrait pas attaquer son ennemi si ces deux-là étaient séparés d'un obstacle. Plus particulièrement, l'eau ne changerait rien, la forêt augmenterait les chances de rater son coup et le tir à travers le rocher serait impossible. Nous aurions vraiment souhaité implémenter cette gestion avec juste les tests des diagonales,

verticales et horizontales (au moins) mais malheureusement le temps aura eu raison de nous.

Néanmoins, notre jeu présente des fonctionnalités dont nous sommes fiers, notamment :

- L'**Intelligence Artificielle** des monstres qui fait que dès que l'un d'entre eux repère un héros, toute leur armée convergera vers celui-là et s'ils en repèrent plusieurs, ils se déplacent vers le plus près. L'IA de base nous paraissait vraiment simpliste et le jeu en devenait facile. Maintenant, elle nous parait assez redoutable et se rapproche plus du raisonnement qu'un humain ferait pour attaquer son ennemi (même si tant que les monstres ne repèrent rien, ils errent un peu sans but, sans deviner où ils doivent aller). Les personnes extérieurs à la Faculté ayant testé le jeu, l'ont testé avant LILAS et après. Ceux-ci ont remarqué que la difficulté était largement accrue après l'implémentation de celle-ci et ainsi le jeu requiert davantage de concentration pour espérer gagner.
- L'**adaptation du « look » du jeu** en fonction du système d'exploitation utilisé.
- La **fluidité du jeu** du fait de l'utilisation du composant graphique. Nous utilisions au départ un GridLayout pour la carte entière, et ainsi à chaque rafraichissement de la carte, chaque case se mettait à jour individuellement. L'effet était plutôt amusant mais pas très rapide et au fur et à mesure que nous rajoutions des images à charger, le jeu devenait beaucoup trop lent. Ainsi, nous avons abandonné l'utilisation du GridLayout et utilisé le composant graphique comme nous le faisions en Travaux Pratiques.
- La gestion d'un maximum d'**erreurs**. Nous avons imaginé le plus de cas possibles dans les actions de l'utilisateur vis-à-vis du jeu, testé un très grand nombre de fois le jeu pour voir toutes les erreurs éventuelles. Nous avons beaucoup modifié le code de la classe Carte en raison d'erreurs de bords de carte, des cas non testés, etc…Par exemple la fonction estIsole a été complètement refaite à un certain moment du projet car elle ne testait pas tous les cas de case du tableau (Première colonne, dernière colonne, première ligne, deuxième ligne, etc..). Nous n'avons à ce jour, plus constaté d'erreurs dans notre jeu.
- La gestion de **sauvegarde et de chargement** de plusieurs fichiers qui stockent le tableau à deux dimensions d'Element. Il a fallu que les classes Element et Position implémentent la classe Serializable pour pouvoir sauvegarder cet objet.

Nos impressions sur le projet ont été que le sujet était très motivant, l'utilisation du Java pour ce type de jeu était plutôt adapté et ne nous a pas demandé tant de temps que ça. Si le jeu devait être programmé en C, nous aurions mis beaucoup plus de temps.

L’utilisation de l’interface graphique était plus attractive qu’en TP car on en faisait une utilisation plus concrète et plus motivante.

Enfin, le projet était selon nous complet, très intéressant et nous a permis d’explorer d’autres aspects du Java (l’entrée/sortie, l’affichage d’images, etc…).