

# RAPPORT L3 INFORMATIQUE

-

Projet de Programmation Orientée Objet

-

BEGGARI – BENGUEZZOU - CHATAIGNIER



# TABLE DES MATIERES

Introduction .....	3
I - Présentation du projet.....	4
1.    Résumé du sujet.....	4
2.    Une première version simpliste du projet.....	4
II - D'une version simpliste à une version complexe .....	5
3.    Merci de patienter .....	5
4.    Cliquer ou ne pas cliquer ? .....	5
5.    Le savant QI de l'IA .....	6
6.    Il est, donc il voit .....	6
7.    Montrez-nous la voie .....	6
8.    Je te dirais qui tu es.....	7
9.    D'une pierre deux coups.....	7
10.   Minia.....	7
11.   L'habit fait le moine.....	8
12.   Le créateur .....	8
13.   Vous avez dit sauvegarde ?.....	8
14.   Home sweet home.....	9
15.   La musique dans la peau.....	9
III - Un problème cache parfois plusieurs solutions .....	10
16.   Le survol d'un ennemi caché .....	10
17.   La gestion des tours .....	10
18.   Glisser, déposer, cliquer, lâcher.....	11
19.   Superposition des couches de dessin.....	11
20.   Les sauvegardes .....	11
21.   La vitesse des Monstre.....	11
Conclusion .....	12
Annexes.....	13
1.    Annexe 1 : Liste exhaustive des ressources utilisées .....	13
2.    Annexe 2 : Schéma descriptif du stockage des actions des Soldats .....	13
3.    Annexe 3 : Schéma descriptif des phases de jeu.....	14
4.    Diagramme UML du package carte .....	15
La classe Carte .....	16
La classe Soldat.....	17

5.	Diagramme UML du package fenetreJeu.....	18
	L'interfaces IConfig .....	19
	La classe PanneauJeu.....	20
6.	Diagramme UML du package menu .....	21
	L'interface IMenu .....	22
7.	Diagramme UML du package music .....	23
8.	Diagramme UML du package utile.....	23
9.	Diagramme UML du package sprite.....	24
	L'interface ISprite .....	25

## INTRODUCTION

L'objectif de ce projet est la création d'un jeu de stratégie en Java permettant de jouer à la guerre dans un monde imaginaire du seigneur des anneaux. Pour concrétiser ce projet, nous avons formé un trinôme, constitué de M. BEGGARI Mohamed Islem, M. BENGUEZZOU Idriss et de M. CHATAIGNIER Antoine.

Tous trois étudiants de licence informatique, nous avons bien entendu certaines notions de programmation et d'algorithmique, mais aucun d'entre nous n'avait découvert le Java de sa propre initiative avant cette troisième année de licence informatique. Ce fût donc une découverte, à la fois du langage, mais aussi des notions de Programmation Orientée Objet indispensables à la conception de projets dans ce langage.

Ce projet n'est pas le premier d'envergure que nous avons eu la possibilité de réaliser au cours de notre licence, mais le travail en groupe induit certaines complications et certaines problématiques inévitables malgré l'expérience. Le monde professionnel ne propose que rarement la possibilité de travailler de façon solitaire, il est donc indispensable pour nous, certainement futurs collaborateurs ou employeurs, de se retrouver face aux différentes problématiques aujourd'hui, pour chercher des solutions utiles à l'avenir.

Nous allons expliciter dans un premier temps le contenu du projet, puis plus bas les différentes fonctionnalités du jeu que nous proposons aujourd'hui. Ce jeu étant très largement inspiré d'autres jeux, nous allons observer les différences et les ressemblances entre les jeux existants et notre jeu, mais nous allons aussi expliquer les problèmes que nous avons rencontré et la façon dont nous les avons abordés, puis résolus.

# I - PRESENTATION DU PROJET

## 1. Résumé du sujet

L'énoncé du projet propose une organisation en classes et interfaces, ainsi que le fonctionnement de quelques méthodes. Cela n'étant qu'une proposition, nous avons tout de même tenu à respecter cet énoncé, en effet, celui-ci nous offrait une base de travail qu'il a été intéressant d'utiliser.

Le sujet impose un type de jeu au tour par tour, avec des Soldats, différenciés en Héros ou en Monstres, eux-mêmes différenciés en différentes races. Ces Soldats se déplacent sur une carte, qu'il faudra redessiner à chaque action, et peuvent rencontrer des Obstacles, qui les empêcheront alors de se mouvoir sur certaines cases. A chaque tour, les Soldats vont effectuer des actions, soit se déplacer, soit attaquer ou se reposer. Un Soldat de type Héros qui se repose gagne des points de vie, mais ce n'est pas le cas pour un Soldat de type Monstre. Les Soldats ont une portée visuelle et une portée de déplacement. La carte n'est pas visible en entier, en effet, seule est visible la zone couverte par la somme des portées visuelle des Soldats de type Héros sur la carte.

Ces éléments (Soldats, Obstacles) vont être représentés graphiquement sur une carte par des images pour certains et des animations (sprites) pour d'autres.

Il est essentiel de pouvoir sauvegarder une partie, de pouvoir la relancer ultérieurement et de pouvoir recommencer une nouvelle partie.

## 2. Une première version simpliste du projet

Après lecture du sujet, et armé des notions élémentaires de Programmation Orientée Objet, il a été évident de concevoir les classes Héros et Monstre comme des classes étendant la classe Soldat. La création d'un JPanel dans une JFrame était elle aussi évidente, mais la première difficulté s'est posée lors des discussions sur la façon de dessiner les éléments sur ce JPanel. Comment dessiner les éléments graphiques ? En les étendant tous de la classe Elément, une classe comportant la fonction seDessiner() permettant de dessiner un Elément sur le JPanel.

Dans sa version simpliste, le jeu est représenté par une liste de cases, certaines colorées pour expliciter la présence de Soldats ou d'Obstacles, et certaines non-colorées pour symboliser la possibilité de se déplacer. Des cases grises représentent la partie non visible de la carte.

## II - D'UNE VERSION SIMPLISTE A UNE VERSION COMPLEXE

La première version de ce projet n'était que la phase 1, c'est à partir de maintenant que l'intérêt s'accroît et que les complications arrivent. Si nous devons nous faire à nous même un reproche, ce serait certainement celui de ne pas avoir assez pensé le projet dans son ensemble avant d'avoir démarré le développement. Une fois la première version terminée, nous avons empilé de nouvelles fonctionnalités les unes après les autres sans vision d'ensemble, et cela nous a causé du tort. Il a fallu résoudre par un autre moyen des problèmes que nous avons déjà résolus, il a fallu casser certains bouts pour reconstruire ensuite.

### 3. Merci de patienter

Ce type de jeu au tour par tour peut se concevoir de deux façons. La première, c'est de laisser le Général-Joueur effectuer une action et de directement la représenter graphiquement à l'écran. Une fois toutes les actions effectuées, le joueur peut finir son tour et c'est aux Monstres de jouer.

Nous avons initialement développé le jeu comme cela, mais avons décidé de changer pour la deuxième façon en s'inspirant d'autres jeux mais aussi parce que cela nous semblait alors plus naturel.

Cette deuxième façon repose sur le fait que le Général-Joueur effectue toutes ses actions mais sans jamais voir le résultat à l'écran. La liste des actions est ensuite exécutée en même temps lorsque le joueur termine son tour. Cela permet d'avoir une multitude d'animations se déroulant en même temps, représentant mieux la réalité d'une troupe en mouvement, offrant ainsi une plus grande immersion au joueur.

Changer de fonctionnement pour offrir la deuxième version au joueur induit logiquement des modifications structurelles de code. La liste des actions doit être dorénavant stockée en mémoire ([cf. annexe 2](#)) et il convient d'apporter une touche visuelle précisant si un Héros a déjà stocké une future action ou s'il est encore disponible. Nous avons opté pour une coloration en rouge de la cellule de la carte sur laquelle se trouve le Héros si celui-ci a déjà une future action prévue.

### 4. Cliquer ou ne pas cliquer ?

A l'origine, le sujet propose d'offrir la possibilité au Général-Joueur de donner ses ordres aux Héros par un glisser-déposer. Ce mode de jeu nous paraissait contre-intuitif de prime abord, nous aurions plutôt imaginé la sélection d'un Héros par un premier clic sur ce dernier, puis l'expression de l'ordre par un second clic sur la carte. Nous avons finalement retenu les deux options, chacune pouvant convenir à un type de joueur différent.

Proposer ces deux modes de jeux n'est pas concevable sans de nouvelles problématiques à résoudre. Il est nécessaire de différencier un clic d'un glisser-déposer, mais une autre question se pose aussi : Comment faire si le Héros est sélectionné et qu'ensuite le Général-Joueur donne son ordre par un glisser-déposer ?

La qualité de jeu a été un élément moteur dans nos décisions tout au long du processus de création. Nous aurions pu nous passer de ce genre de problématiques mais cela aurait signifié se fermer à nos volontés et ne pas essayer de surmonter les obstacles, or, ce sont justement les obstacles qui nous font et nous feront progresser.

## 5. Le savant QI de l'IA

multiples sont les jeux proposant à un joueur solitaire d'affronter la puissance de calcul d'un ordinateur, l'un contre l'autre. Notre jeu proposant des ennemis, représenté par la classe Monstre, il a fallu trouver un moyen de faire effectuer des actions logiques à ces derniers. Certes, nous aurions pu nous contenter d'une variable aléatoire pour définir l'action du Monstre, entre se déplacer pour fuir, se déplacer pour se rapprocher d'un Héros, attaquer un Héros ou ne rien faire mais cela aurait fortement affaibli la qualité de jeu.

Nous proposons donc une fonction permettant au monstre soit de fuir s'il est trop proche d'un Héros ou si son nombre de points de vie atteint un seuil critique, soit de se rapprocher s'il est trop loin pour attaquer le Héros. Le Général-Joueur se retrouve donc à jouer contre des ennemis réfléchis, capables de modifier leurs comportements face à la réalité du jeu.

## 6. Il est, donc il voit

Chaque entité dispose d'une vision qui lui est propre, c'est notamment ce qui permet à la carte de se dévoiler au Général-Joueur selon les emplacements de ses Héros. Il est indispensable de représenter cette vision à l'écran par une coloration des cases symbolisant la vision du Héros lors de la sélection de celui-ci par exemple. C'est l'option que nous avons choisie, cette représentation étant utilisée couramment dans de nombreux jeux. Une fois la carte dessinée et recouverte de case la masquant, la liste des Héros est parcourue et pour chaque Héros, les cases dans son champ de vision sont alors dévoilées au joueur.

On identifiera ici qu'une optimisation aurait pu être trouvée avec plus de temps. En effet, si deux Héros partagent des cases de la carte dans leurs champs de visions respectifs, ces cases seront dévoilées à deux reprises, une fois pour chaque Héros.

La réflexion pourrait s'arrêter ici, mais les Héros ne sont pas uniquement des paires d'yeux doués de portée, ils sont aussi des marathoniens capables de se déplacer sur la carte. A ce titre, il est important de représenter les cases sur lesquels le Héros peut se mouvoir en plus de celles qu'il peut voir. Nous avons simplement décidé de colorier de façon plus foncée (moins transparente) les cases adjacentes au Héros disponibles au déplacement.

## 7. Montrez-nous la voie

Conservant notre volonté de rendre le jeu ludique et agréable d'utilisation pour tous, nous avons eu l'idée de représenter à l'avance l'impact d'un déplacement d'un Héros sur la carte. Cela signifie que lorsqu'un Héros est sélectionné, ou que le joueur est en phase de déplacement avec un glisser-déposer, une copie du Héros va être générée sur la case de la carte survolée par le joueur. Cela permet, en plus de générer une animation supplémentaire

agréable au jeu, de représenter visuellement de façon immédiate les modifications du plateau de jeu, évitant par exemple une erreur ou un déplacement sur une mauvaise case.

Nous avons aussi souhaité apporter une touche supplémentaire à ces fonctionnalités, à savoir la modification de l'animation (sprite) du Héros sélectionné lorsque la souris tourne autour du Héros pendant une phase de placement. En des termes plus clairs, une fois sélectionnée ou pendant une phase de glisser-déposer sur une case adjacente, le Héros se tourne à gauche si la souris est à gauche, en bas si la souris est en bas, etc...

## 8. Je te dirais qui tu es

Parlons maintenant d'une fonctionnalité plus simple à mettre en œuvre : l'histoire des entités.

La presque totalité des jeux existants intègrent une histoire, un univers qui leur est propre permettant au joueur de se détacher de la réalité mais permettant aussi de servir le design ou d'imposer une certaine cohérence d'ensemble. Plus cette histoire est développée, et plus cela permettra la suspension consentie de l'incrédulité du joueur, c'est-à-dire l'acceptation, le temps du jeu, de l'existence d'un monde imaginaire peuplé d'orcs et d'elfes.

## 9. D'une pierre deux coups

Cette fonctionnalité est à la fois essentielle au bon fonctionnement du jeu et en même temps très délicate à mettre en place : la gestion des projectiles.

La version que nous proposons aujourd'hui est une gestion simpliste des projectiles. Celui-ci est expédié par un Soldat vers un autre Soldat, il suffit alors de créer une trajectoire sous forme de ligne entre les deux Soldats.

L'ambition aurait été de générer une trajectoire sous forme parabolique, avec un contournement des obstacles si nécessaire permettant d'apporter une touche de réalisme mais nous avons préféré conserver cette solution fonctionnelle et parfaitement acceptable pour consacrer notre temps aux problèmes restants.

L'image représentative d'un projectile est elle aussi restée simpliste. Même si des archers sont présents sur la carte, ce sont toujours des cailloux qui sont échangés, cela pourra être modifié rapidement à l'avenir si des joueurs le demandent.

## 10. Minia

De nombreux jeux de plateau proposent au joueur une carte de taille supérieure à la zone affichée à l'écran, offrant un monde plus vaste demandant au joueur d'agir sur plusieurs fronts en même temps. Pour se repérer dans le monde, une mini-carte est souvent située en haut à droite de l'écran de jeu avec une démarcation de la zone actuellement visible à l'écran. Nous avons repris ce concept, avec la création d'une mini-carte, mais aussi avec la possibilité de se déplacer rapidement d'une zone à une autre en cliquant sur cette dernière.



Nous ne cessons de le répéter, mais encore une fois, cette implémentation est née de la volonté d'augmenter le plaisir ressenti du Général-Joueur jouant à notre jeu.

Cette amélioration semble évidente d'utilisation pour les joueurs expérimentés, mais pour les joueurs non accoutumés de ce type de jeu, nous avons décidé de proposer deux autres alternatives. Ces derniers pourront se déplacer sur la carte grâce à des flèches (haut, bas, gauche et droite) qui permettent de déplacer la zone visible d'une case dans les quatre directions principales ou en cliquant sur une case vide, puis en déplaçant la souris en bas (pour se déplacer en haut), en haut (pour se déplacer en bas), à gauche (pour aller à droite) etc.. dans toutes les directions possibles.

## 11. L'habit fait le moine

Le jeu, dans sa version simpliste, proposait une coloration adéquate des cases de la carte pour représenter la présence de Héros, de Monstres ou d'Obstacles. Notre deuxième version proposée pour ce jeu préférerait à cela la génération d'images représentatives des entités, mais la dernière version intègre aujourd'hui des animations de Héros ou de Monstres comme représentation graphique à l'écran. Bien entendu, il est nécessaire de modifier ces animations selon l'action du Soldat, que ce soit un déplacement ou une attaque. Il nous a aussi paru essentiel d'apporter une touche de réalisme en tournant le Soldat selon sa direction ou sa cible, le représentant parfois de dos, parfois de profil, ou parfois de face.

## 12. Le créateur

La possibilité de personnaliser un jeu immerge encore un peu plus les joueurs dans l'univers, c'est pourquoi nous proposons à chacun de concevoir sa propre carte à travers un nouveau mode de jeu. Une fois les obstacles et les soldats posés aux positions de son choix, le Général-Joueur peut alors lancer sa partie avec la carte nouvellement créée. Cela permet de concevoir des cartes atypiques, de proposer des défis et de les partager à d'autres joueurs à travers un échange de sauvegardes.

## 13. Vous avez dit sauvegarde ?

Une partie des jeux de type tour par tour peut parfois durer un certain temps, et le joueur peut être amené à quitter la partie en voulant la continuer plus tard. C'est pourquoi nous offrons la possibilité à chacun de créer une sauvegarde pour la lire plus tard. Plusieurs sauvegardes peuvent être stockées, permettant à chacun de conserver ses meilleures parties. La création de sauvegarde implique alors la nécessité d'offrir la possibilité à l'utilisateur de supprimer des anciennes sauvegardes pour nettoyer son historique et ne garder que les parties qui l'intéressent. Pour ce faire, nous proposons une icône explicite de corbeille activant ou désactivant la suppression de sauvegarde.

En parlant de sauvegarde, nous sous-entendons la sauvegarde d'une carte puis la régénération de celle-ci à partir d'un fichier source. Il a donc fallu créer des fonctions pour quitter le jeu et le relancer à partir d'une carte vide ou d'une sauvegarde précédente.

## 14. Home sweet home

Au lancement du programme, toutes les actions disponibles doivent être proposées à l'écran, que ce soit le lancement d'une nouvelle partie, la lecture d'une sauvegarde, la création d'une carte ou même la possibilité de quitter le jeu après avoir cliqué sur l'icône par erreur. A ce titre, le menu est un élément indispensable du jeu, il sert d'interface entre l'utilisateur et les différentes options proposées par le jeu. C'est aussi la première image visible par le futur Général-Joueur, il faut alors que le menu exprime d'un simple coup d'œil la nature du jeu, l'objectif, l'histoire et l'univers, il doit immerger le joueur dès le lancement du programme.

Pour immerger notre joueur, nous avons décidé de mettre en arrière-plan une image scindée en deux parties : nous retrouvons à gauche un Héros, et à droite un Monstre, les deux semblant sur le point de s'affronter pour gagner.

On retrouve sur ce menu les principales actions proposées avec différents boutons qu'il n'est plus nécessaire de présenter tellement ils sont présents sur l'intégralité des jeux depuis 1980.

## 15. La musique dans la peau

La dernière fonctionnalité que nous présenterons ici est l'intégration de musique dans le jeu. Le son n'est pas un élément indispensable aux jeux mais il permet de faire pénétrer le joueur dans une ambiance, dans un univers choisi par les créateurs, son but est immersif. Nous avons donc intégré sur le Menu et dans le jeu des musiques permettant de dynamiser le jeu.

En conclusion, nous pouvons dire que l'intérêt de la création d'un tel jeu réside dans l'ajout de fonctionnalités, toujours un peu plus ambitieuses. La version simpliste fût intéressante à créer mais l'apport de nouveaux éléments permet de faire croître sans arrêt le plaisir du futur Général-Joueur. Nous avons un certain nombre d'idées à la découverte du sujet, mais c'est en prenant nous-même le rôle de Général-Joueurs que de nouvelles idées d'amélioration ont émergé.

### III - UN PROBLEME CACHE PARFOIS PLUSIEURS SOLUTIONS

Durant la conception du jeu, nous avons rencontré un grand nombre de problématiques à résoudre, voici une présentation d'une partie d'entre elles.

#### 16. Le survol d'un ennemi caché

La zone de la carte visible à l'écran n'est pas forcément dévoilée au Général-Joueur dans son intégralité. Si une cellule de la carte n'entre dans aucune portée de visibilité des Héros, alors celle-ci reste masquée aux yeux du joueurs. Le problème qui s'est alors posé est le suivant : Comment masquer les informations du Soldat survolé si ce dernier est masqué ? Ou plus précisément, comment ne pas dévoiler la présence d'un Monstre lors d'un survol de la souris si ce dernier se trouve en dehors de la zone visible ?

Pour résoudre ce problème, nous avons trouvé une solution, certes assez peu optimisée, mais opérationnelle. Celle-ci consiste à contrôler, à chaque mouvement de la souris la cellule survolée, puis à analyser pour chaque Héros présent sur la carte l'existence ou non de la cellule dans sa portée visuelle.

On devine aisément ici le problème de performance si nous retrouvons un très grand nombre de Héros sur la carte et que la souris passe sur une zone non-visible. Il faudra alors parcourir l'ensemble de la liste des Héros. Cependant, nous avons conservé cette solution car le nombre de Héros présent n'est jamais trop important pour un ordinateur, nous ne travaillons pas sur des cartes assez grandes pour remarquer une différence notable.

Une idée d'optimisation pourrait être de stocker dans une liste les cellules visibles sur la carte, puis de mettre à jour, de façon intéressante, cette liste à chaque déplacement de Héros.

#### 17. La gestion des tours

La durée des animations à la fin d'un tour peut être importante, les héros vont en effet se déplacer ou éventuellement s'attaquer entre eux. Le Général-Joueur peut avoir envie de jouer immédiatement après, mais il se rendra compte, à la suite d'un clic, que cela n'est pas possible. Tant que les Soldats n'ont pas fini leurs actions, il est en effet impossible de sélectionner un Héros ou de préparer son futur tour, même s'il reste une action en dehors du champ visible de la carte, à l'autre bout du monde. Pour résoudre ce problème de jeu, nous avons mis en place une coloration des cases si le Héros est en action, permettant de savoir d'un simple coup d'œil si le Général-Joueur peut jouer de nouveau.

Ce problème en révèle un autre de taille : Que se passe-t-il si le joueur clique sans cesse sur le bouton "Fin de tour" alors que les animations ne sont pas terminées ? Il a fallu donc ici aussi mettre en place une stratégie pour désactiver certains boutons pendant la phase d'animation des tours.

Enfin, le dernier problème qui s'est posé avec la gestion des tours a été un problème structurel. Comment mémoriser les différentes actions des Soldats ? Comment gérer les

déplacements ou les attaques en même temps ? Comment retenir le Soldat attaqué et le soldat attaquant alors qu'aucun des deux n'est plus sélectionné ?

Pour cela, nous avons mis en place des listes, notamment listeActionAttaque et listeActionDeplacement, deux listes de Soldats. La première stocke, pour chaque attaque, le Soldat attaquant et le Soldat attaqué, cela permet de connaître, en parcourant les index pairs de la liste, de connaître les attaquant, et, en parcourant les index impairs, les attaqués.

La liste listeActionDeplacement fonctionne légèrement différemment, elle va stocker en mémoire le Soldat avant son déplacement, et un clone du Soldat après son déplacement. Cela évite de retrouver deux Soldats sur la même case.

## 18. Glisser, déposer, cliquer, lâcher

La gestion des événements de la souris fût, de manière générale, complexe à mettre en œuvre. Nous proposons ici une multitude d'actions différentes, autorisant une clique ou un glisser-déposer pour une même action, et certaine fois un mélange des deux pour une meilleure expérience utilisateur. Cette liberté d'action demande de garder en mémoire les cliques précédentes, d'analyser à chaque mouvement de la souris si le clique est actif, si un Héros est sélectionné et une multitude d'autres facteurs impactant l'action en cours.

## 19. Superposition des couches de dessin

Les différents niveaux de dessins ont aussi soulevé des problématiques. Lorsqu'un Héros se déplace d'une case à une autre, ce dernier ne bouge pas réellement d'une cellule à l'autre tant que l'animation n'est pas terminée. C'est pourquoi il disparaissait sous les cases du terrain une fois que son animation (sprite) sortait de sa case. Pour résoudre ce problème, nous avons dû scinder le dessin des Héros en deux boucles, la première pour dessiner les zones visibles de la carte, avec des images de terre, et la seconde pour dessiner les animations des Héros, permettant de les passer au premier plan.

## 20. Les sauvegardes

La mise en place de sauvegardes nous a posé des problèmes, notamment quand nous voulions concaténer des chaînes de caractères pour définir l'emplacement de la sauvegarde. Pour résoudre ce problème, nous avons dû supprimer la concaténation et ne définir le chemin qu'avec une simple chaîne de caractères.

## 21. La vitesse des Monstre

Chaque Monstre est dessiné par un Héros s'il se trouve dans son champ de vision, mais cela génère des problèmes lorsque deux Héros partagent une cellule dans leurs champs de vision respectifs et qu'un Monstre se trouve sur cette cellule. Les deux Héros vont lancer l'animation de déplacement du Monstre, ce qui le fera bouger deux fois plus vite qu'à l'accoutumée. Ce problème peut être résolu en stockant par exemple une variable pour chaque Monstre afin de déterminer si son déplacement a déjà été initié, évitant ainsi cette double boucle.

Nous n'avons pas mis en place cette correction, car nous avons trouvé que faire se mouvoir le Monstre X fois plus vite lorsque X Héros sont proches de lui apportait une touche d'humour, donnant l'impression que le Monstre se réfugie plus vite loin de ses ennemis lorsque ceux-ci arrivent en nombre.

## CONCLUSION

Ce projet, dans lequel nous nous sommes investis ce semestre, nous a apporté un nombre important de connaissances en matière Programmation Orientée Objet. Aucun de nous ne connaissait le langage Java il y a encore quelques mois. Nous connaissions son nom, son utilité, sa praticité, mais c'est avec ce projet que les différents concepts de POO se sont imprimés dans notre esprit.

Résoudre des problématiques d'algorithmique et de programmation en développant un jeu vidéo fût une expérience très enrichissante en plus d'être ludique. Travailler en groupe n'est pas toujours facile, bien au contraire, la distance demande une organisation conséquente. Nous avons heureusement pu nous retrouver suffisamment souvent pour faire des points ensemble, mettre à jour nos idées respectives, et travailler dans la même pièce.

Un tel projet n'est jamais réellement terminé, il n'est jamais possible d'affirmer qu'il n'y a plus rien à faire, car il y aura toujours une nouvelle fonctionnalité à apporter, une autre à revoir ou une fonction à améliorer. Cependant, nous sommes très heureux de partager aujourd'hui une version fonctionnelle de WarStone dans laquelle nous avons mis beaucoup de notre énergie et de notre temps.

# ANNEXES

## 1. Annexe 1 : Liste exhaustive des ressources utilisées

Pour résoudre une grande partie des problèmes rencontrés :

PDFs de cours

Pour les problèmes restants rencontrés :

<https://stackoverflow.com/>

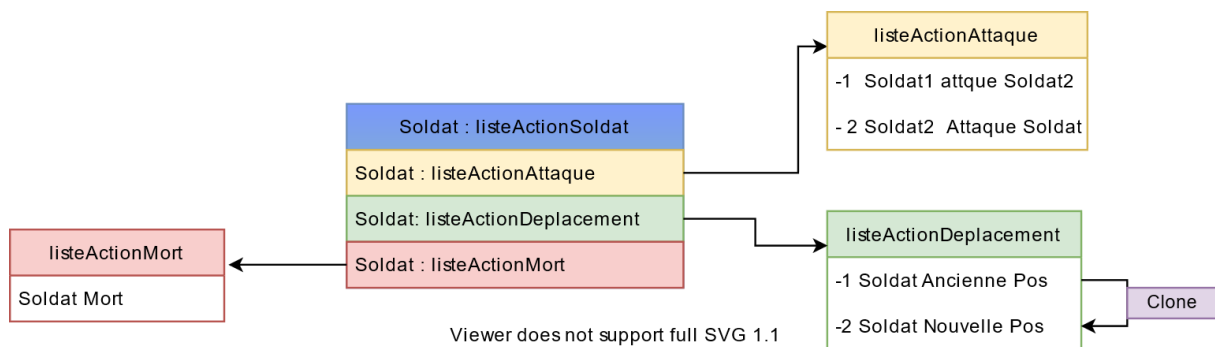
Pour la génération de la flèche de déplacement sur la carte

<https://stackoverflow.com/questions/2027613/how-to-draw-a-directed-arrow-line-in-java>

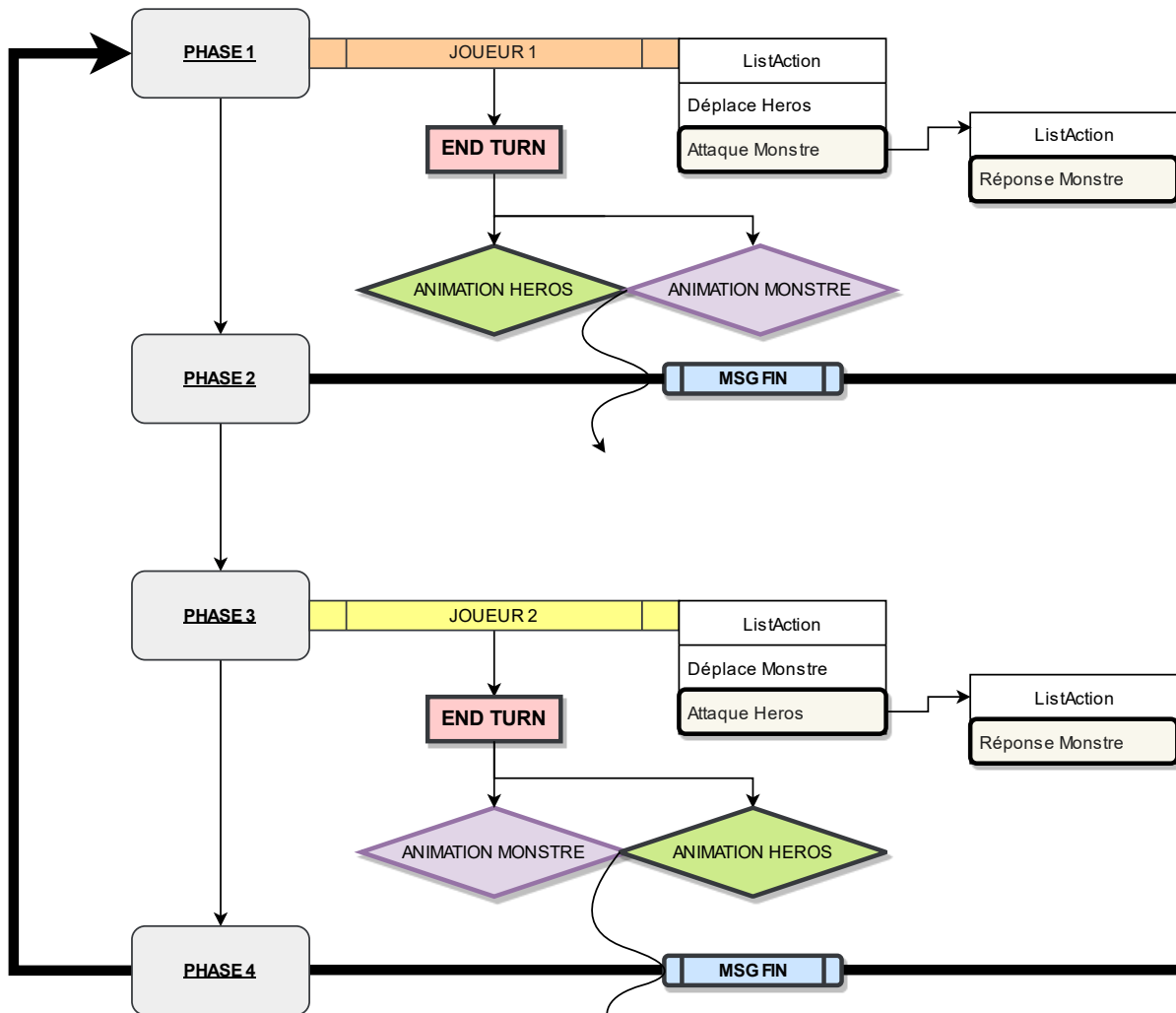
Pour comprendre les dépendances et le fonctionnement de certaines fonctions :

<https://www.java.com/fr/>

## 2. Annexe 2 : Schéma descriptif du stockage des actions des Soldats



### 3. Annexe 3 : Schéma descriptif des phases de jeu



Viewer does not support full SVG 1.1

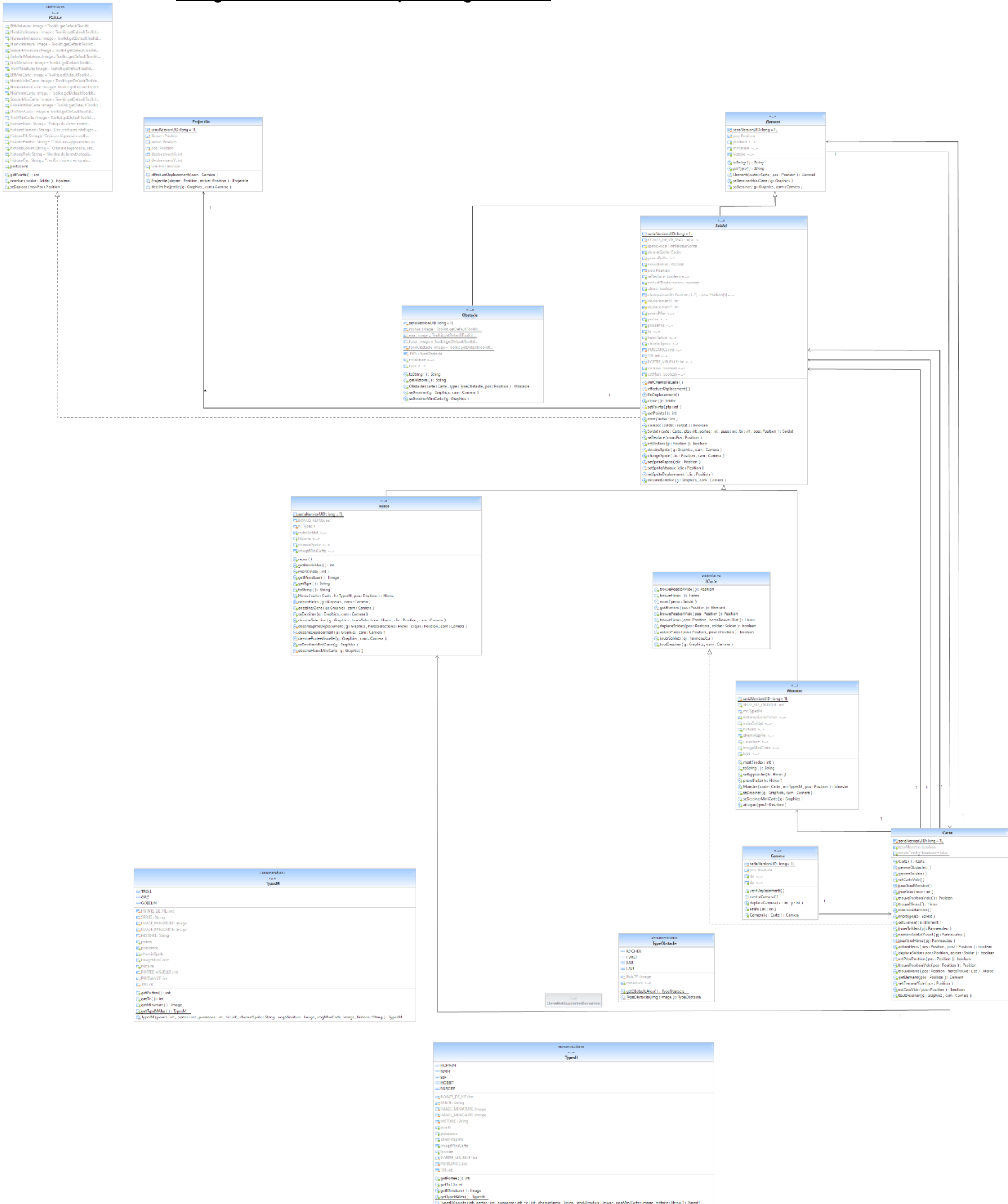
Le schéma ci-dessus représente les différentes phases de jeu.

En phase 1, le joueur effectue une liste d’actions, chaque déplacement insère le Soldat et un clone de ce Soldat dans listeActionDéplacement, chaque attaque insère le Soldat attaquant, le Soldat attaqué, puis son opposé (la riposte), c’est-à-dire le Soldat attaqué (devenu attaquant) et le Soldat attaquant (devenu attaqué).

A la fin du tour, les animations sont lancées et un message informe de la fin de ces animations. Dans WarStone, le message de fin d’animation est le retour à la normale des colorations des cases.



























En phase 3, c’est le joueur opposé qui joue, ici l’ordinateur, et la logique reste la même. S’ensuit la phase d’animation, puis un retour en phase 1.

# 4. Diagramme UML du package carte



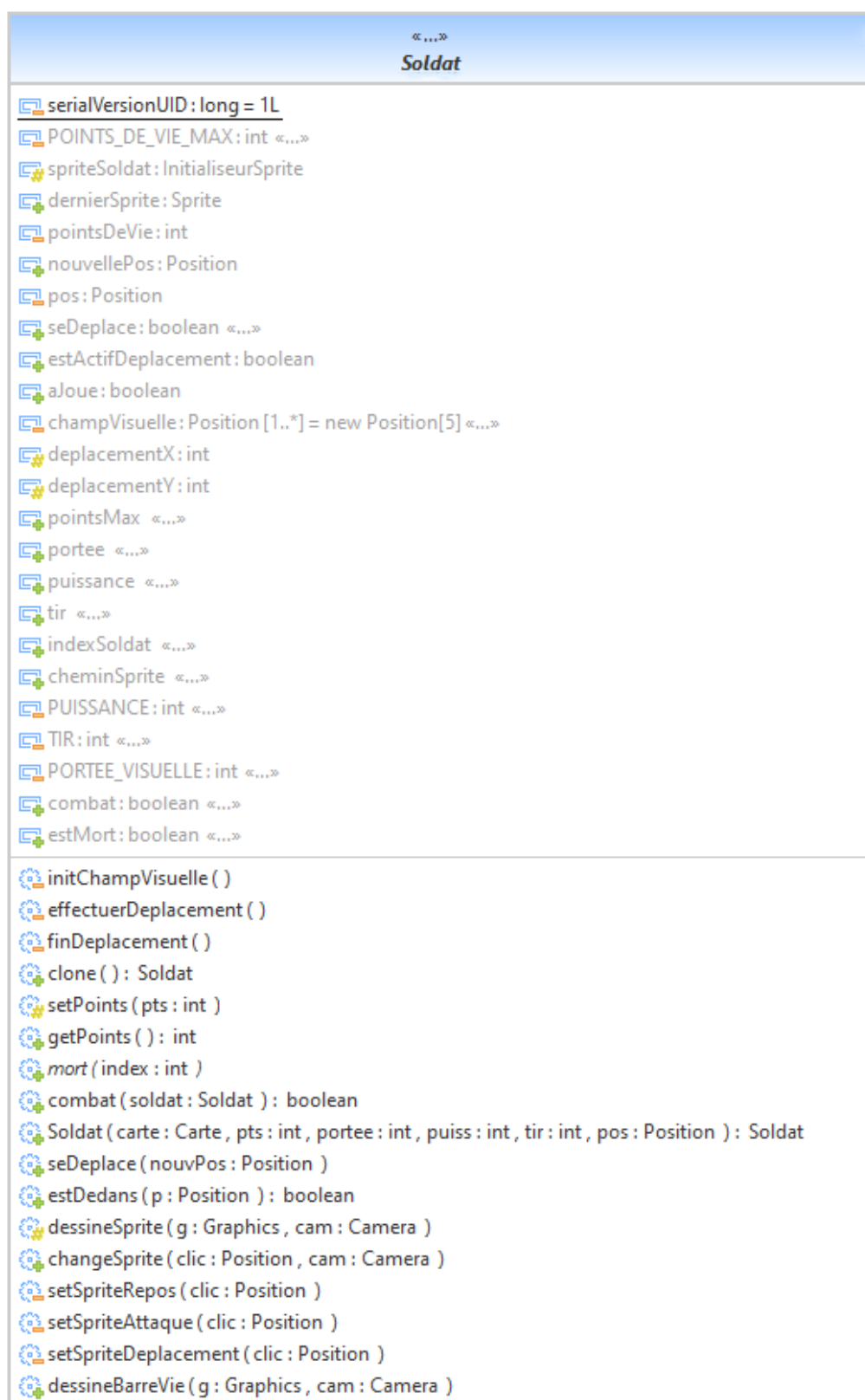


## LA CLASSE CARTE

Carte
 serialVersionUID : long = 1L
 tourMonstre : boolean
 modeConfig : boolean = false
 Carte ( ) : Carte
 genereObstacles ( )
 genereSoldats ( )
 setCarteVide ( )
 joueTourMonstre ( )
 joueTour ( tour : int )
 trouvePositionVide ( ) : Position
 trouveHeros ( ) : Heros
 removeAllAction ( )
 mort ( perso : Soldat )
 setElement ( e : Element )
 jouerSoldats ( pj : PanneauJeu )
 nombreSoldatVivant ( pj : PanneauJeu )
 joueTourHeros ( pj : PanneauJeu )
 actionHeros ( pos : Position , pos2 : Position ) : boolean
 deplaceSoldat ( pos : Position , soldat : Soldat ) : boolean
 estPrisePosition ( pos : Position ) : boolean
 trouvePositionVide ( pos : Position ) : Position
 trouveHeros ( pos : Position , herosTrouve : List ) : Heros
 getElement ( pos : Position ) : Element
 setElementVide ( pos : Position )
 estCaseVide ( pos : Position ) : boolean
 toutDessiner ( g : Graphics , cam : Camera )

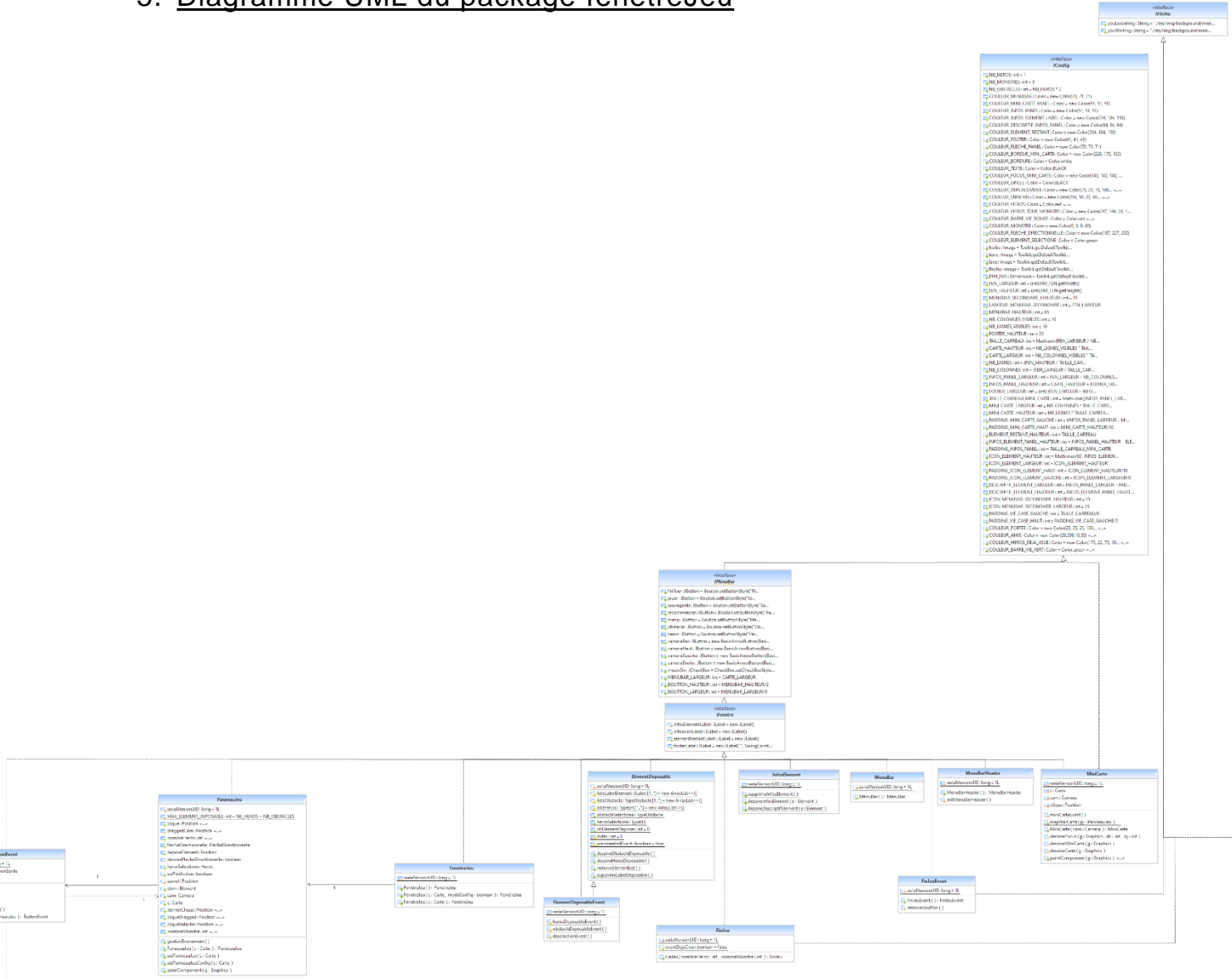
La classe Carte comporte les fonctions utiles par exemple à la génération de nouveaux Soldats ou encore à la récupération d'informations, comme le nombre de Soldats vivants, l'occupation ou non d'une case par un Soldat ou par un obstacle.

## LA CLASSE SOLDAT

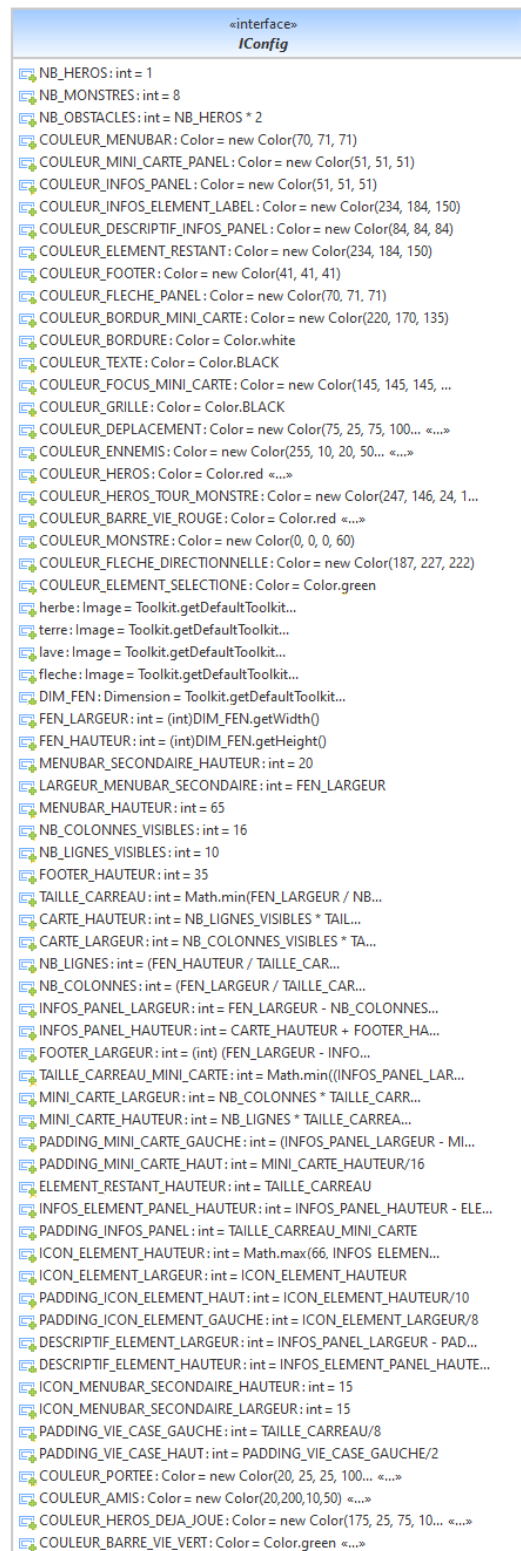


La classe Soldat est de taille plus conséquente encore car le nombre de variables à conserver est plus important que pour la Carte. La classe contient les fonctions de dessin, de combat ou encore de clonage.

### 5. Diagramme UML du package fenetreJeu

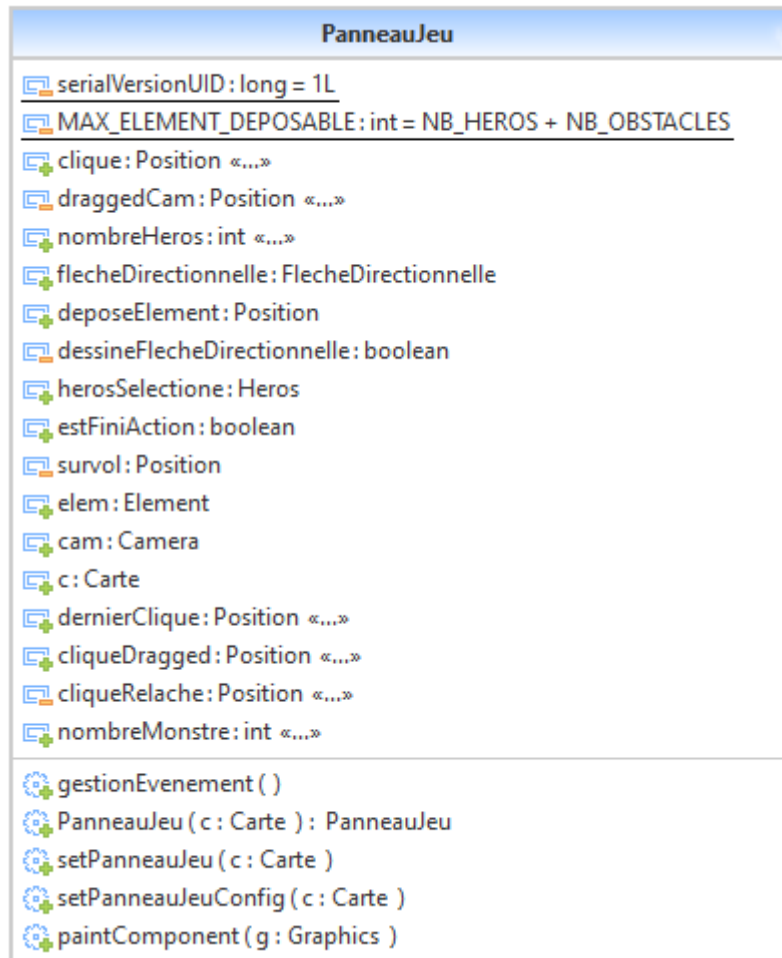


## L'INTERFACES ICONFIG



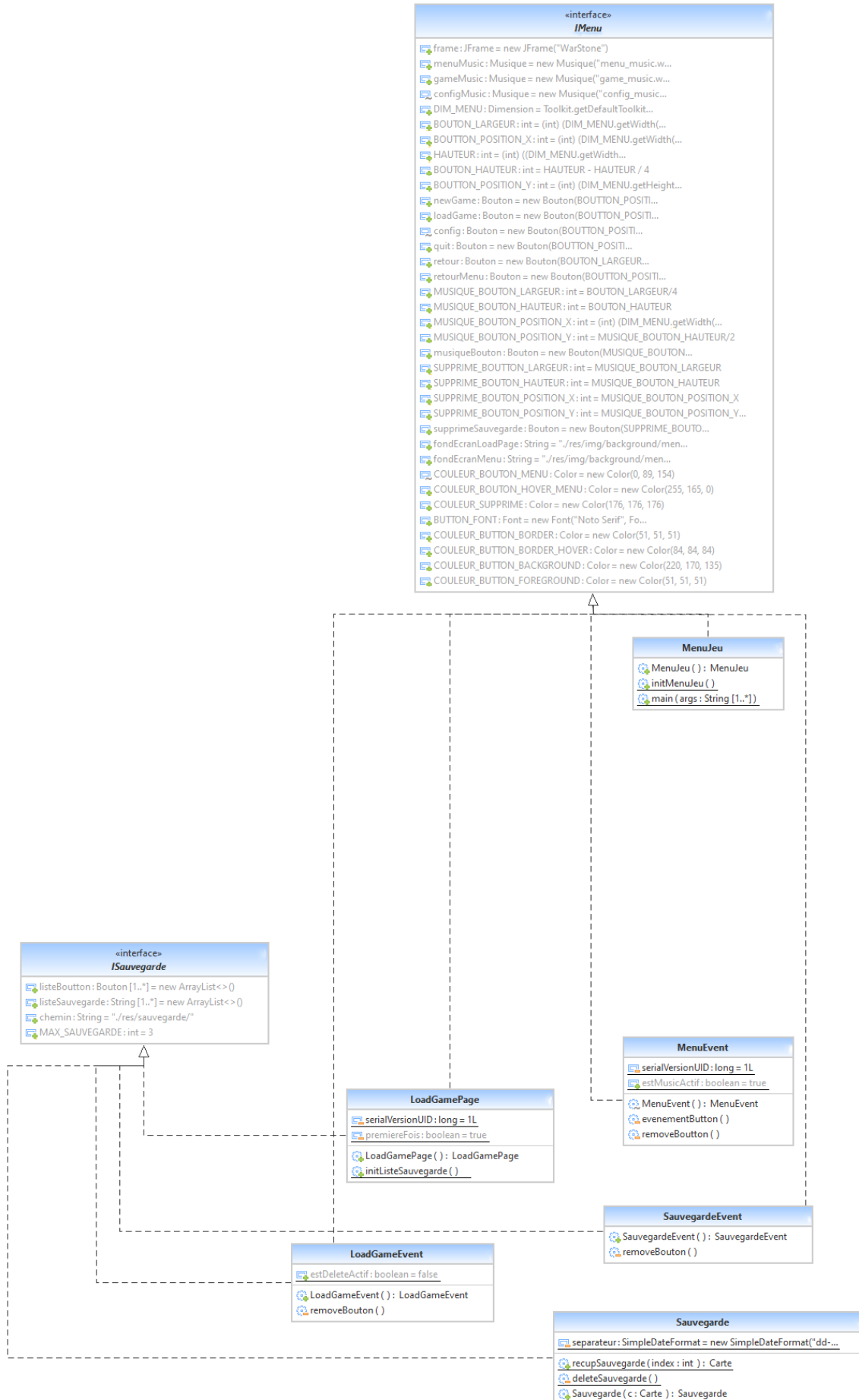
Cette interface va stocker en mémoire les variables de dimension des différents panneaux de jeux, les espaces, les couleurs, les images etc...

## LA CLASSE PANNEAUJEU



Cette classe est une classe centrale du projet, c'est elle qui va gérer les événements de l'utilisateur sur le panneau et de façon plus générale, le lien entre le Général-Joueur et le programme.

## 6. Diagramme UML du package menu



## L'INTERFACE IMENU

```
IMenu
frame: JFrame = new JFrame("WarStone")
menuMusic: Musique = new Musique("menu_music.w...
gameMusic: Musique = new Musique("game_music.w...
configMusic: Musique = new Musique("config_music...
DIM_MENU: Dimension = Toolkit.getDefaultToolkit...
BOUTON_LARGEUR: int = (int) (DIM_MENU.getWidth(...
BOUTON_POSITION_X: int = (int) (DIM_MENU.getWidth(...
HAUTEUR: int = (int) ((DIM_MENU.getWidth...
BOUTON_HAUTEUR: int = HAUTEUR - HAUTEUR / 4
BOUTON_POSITION_Y: int = (int) (DIM_MENU.getHeight...
newGame: Bouton = new Bouton(BOUTON_POSITI...
loadGame: Bouton = new Bouton(BOUTON_POSITI...
config: Bouton = new Bouton(BOUTON_POSITI...
quit: Bouton = new Bouton(BOUTON_POSITI...
retour: Bouton = new Bouton(BOUTON_LARGEUR...
retourMenu: Bouton = new Bouton(BOUTON_POSITI...
MUSIQUE_BOUTON_LARGEUR: int = BOUTON_LARGEUR/4
MUSIQUE_BOUTON_HAUTEUR: int = BOUTON_HAUTEUR
MUSIQUE_BOUTON_POSITION_X: int = (int) (DIM_MENU.getWidth(...
MUSIQUE_BOUTON_POSITION_Y: int = MUSIQUE_BOUTON_HAUTEUR/2
musiqueBouton: Bouton = new Bouton(MUSIQUE_BOUTON...
SUPPRIME_BOUTON_LARGEUR: int = MUSIQUE_BOUTON_LARGEUR
SUPPRIME_BOUTON_HAUTEUR: int = MUSIQUE_BOUTON_HAUTEUR
SUPPRIME_BOUTON_POSITION_X: int = MUSIQUE_BOUTON_POSITION_X
SUPPRIME_BOUTON_POSITION_Y: int = MUSIQUE_BOUTON_POSITION_Y...
supprimeSauvegarde: Bouton = new Bouton(SUPPRIME_BOUTO...
fondEcranLoadPage: String = "./res/img/background/men...
fondEcranMenu: String = "./res/img/background/men...
COULEUR_BOUTON_MENU: Color = new Color(0, 89, 154)
COULEUR_BOUTON_HOVER_MENU: Color = new Color(255, 165, 0)
COULEUR_SUPPRIME: Color = new Color(176, 176, 176)
BUTTON_FONT: Font = new Font("Noto Serif", Fo...
COULEUR_BUTTON_BORDER: Color = new Color(51, 51, 51)
COULEUR_BUTTON_BORDER_HOVER: Color = new Color(84, 84, 84)
COULEUR_BUTTON_BACKGROUND: Color = new Color(220, 170, 135)
```

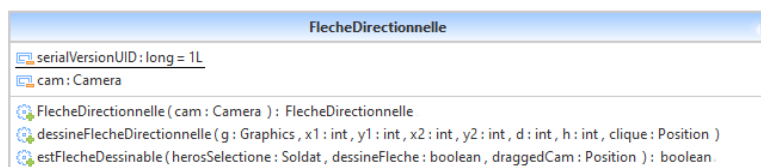
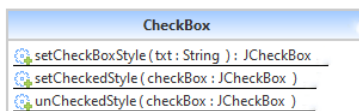
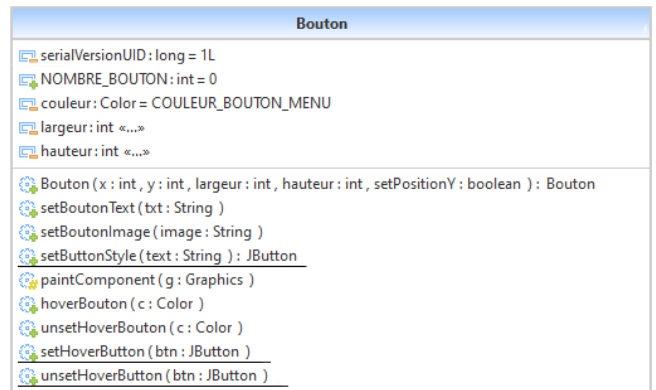
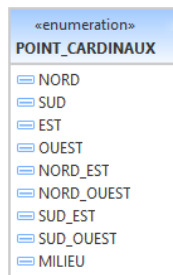
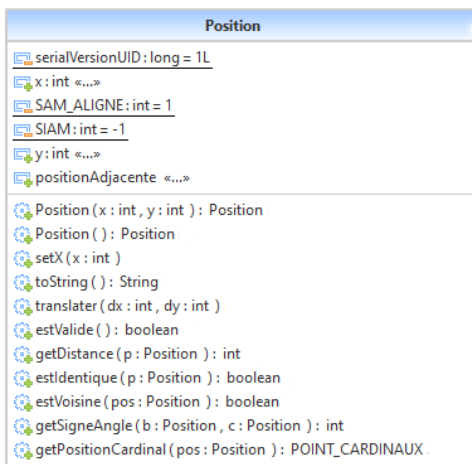
Comme l'interface IConfig, celle-ci permet de stocker en mémoire les couleurs, les images et les dimensions

## 7. Diagramme UML du package music



Ce package est utile pour la gestion de la musique.

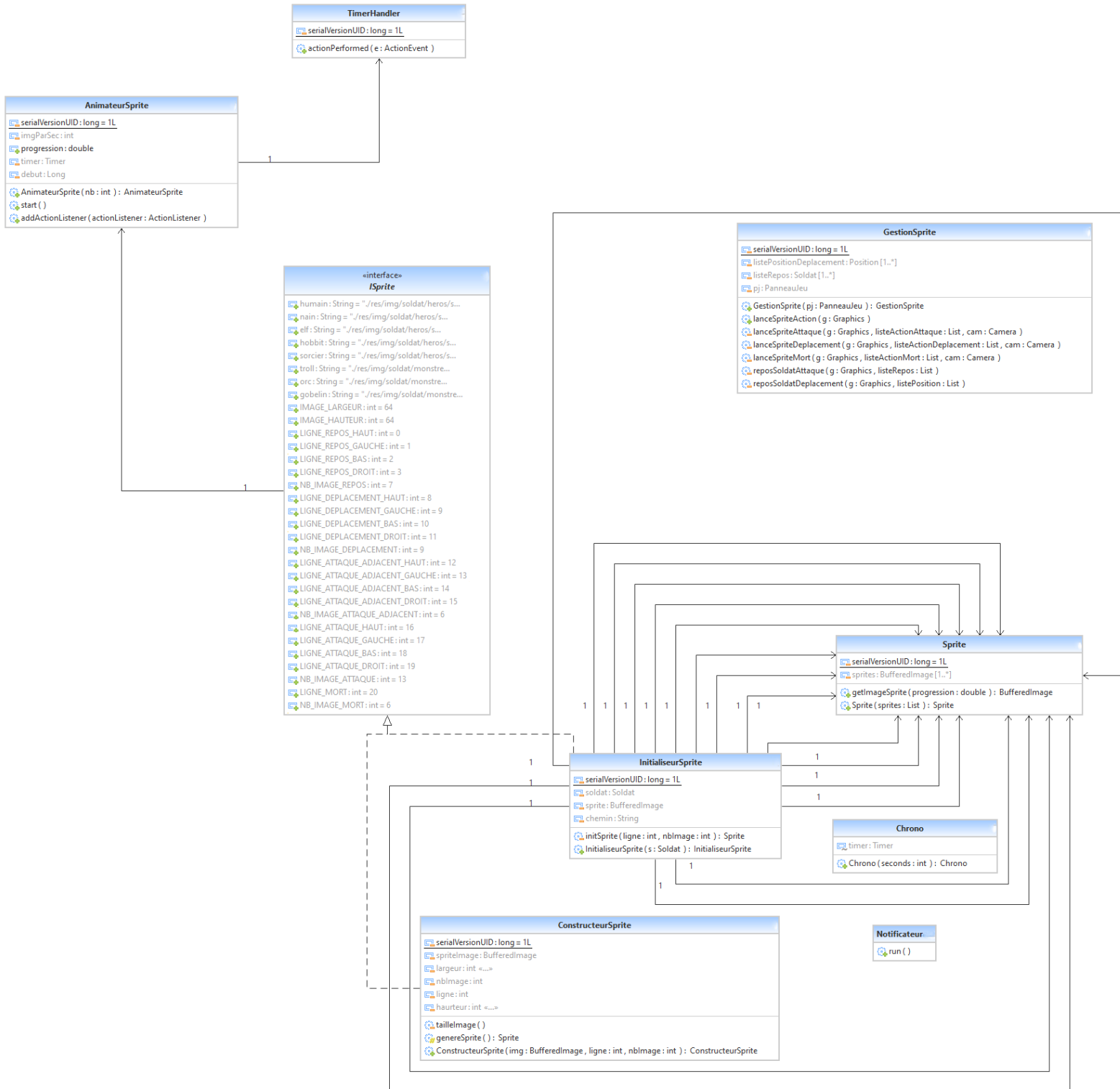
## 8. Diagramme UML du package utile



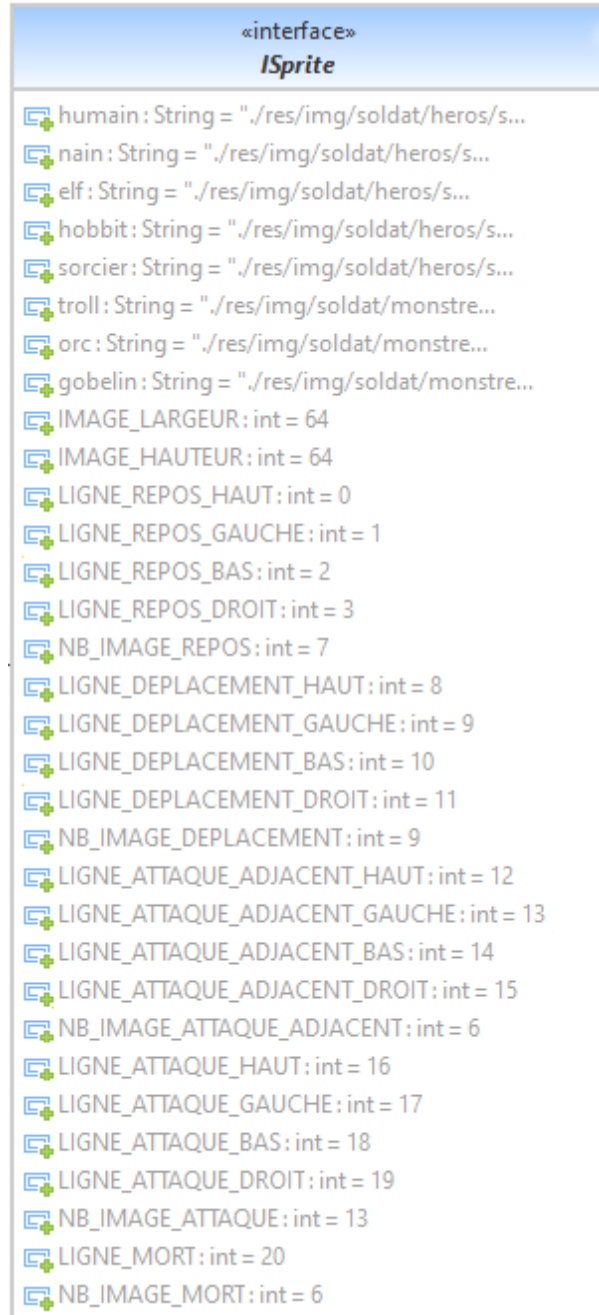
Le package utile est utilisé pour différentes classes utiles au programme, comme la classe Position, Bouton ou la classe permettant de contrôler les flèches directionnelles.



## 9. Diagramme UML du package sprite



## L'INTERFACE ISPRITE



Cette interface ISprite est utile pour la gestion des sprites. On stocke ici les chemins vers les images de sprites, les dimensions ainsi que le nombre d'images pour un déplacement ou bien une attaque.