|  |
| --- |
| Projet Concept Objet |
| La communauté de l’anneau |

|  |
| --- |
| Jules Guiot  Benjamin Legrand  Zoé Moulart  Clément Villalba  24/10/2019 |

Table des matières

[Introduction 2](#_Toc24110858)

[Description du projet 3](#_Toc24110859)

[Nos choix 3](#_Toc24110860)

[Individus 3](#_Toc24110861)

[Carte 3](#_Toc24110862)

[Déplacements 4](#_Toc24110863)

[Rencontres 4](#_Toc24110864)

[Combat 4](#_Toc24110865)

[Déroulement d’une partie 4](#_Toc24110866)

[Description du code 5](#_Toc24110867)

[Les 3 concepts fondamentaux 5](#_Toc24110868)

[Singleton 5](#_Toc24110869)

[Variables statiques 5](#_Toc24110870)

[Nombre pseudo-aléatoire 5](#_Toc24110871)

[Diagrammes UML 6](#_Toc24110872)

[Diagramme de classes 6](#_Toc24110873)

[Diagramme d’états-transitions 7](#_Toc24110874)

[Diagramme d’activités 10](#_Toc24110875)

[Gestion du projet 12](#_Toc24110876)

[Conclusion 12](#_Toc24110877)

# Introduction

Ce document est un rapport du projet qu’il nous a été demandé de réaliser dans le cadre du cours de concept objet.

Dans un premier temps nous dresseront une description du projet en expliquant nos choix et le fonctionnement de la simulation.

Ensuite nous présenteront les éléments que nous avons intégré au code pour répondre aux attentes des consignes (3 principes fondamentaux, singletons, classes statiques etc.) et les diagrammes UML qu’il nous a été demandé de réaliser.

Enfin nous expliqueront comment s’est déroulé la gestion de ce projet.

# Description du projet

Pour ce projet nous avons décidé de suivre l’exemple décrit dans les consignes présentant le projet, en adaptant notre projet pour les différentes zones d’ombre du document.

## Nos choix

Dans cette partie nous présenterons nos choix pour ce projet, certains correspondent à ceux des consignes.

### Individus

Dans notre projet, deux alliances s’affronterons, les gentils et les méchants divisées en 2 populations chacune : Les humains et les elfes pour les gentils, les orcs et les gobelins pour les méchants.

Ces populations sont composées d’un nombre d’individus pseudo-aléatoires :

* Humain = 1 chance sur 6
* Elfe = 1 chance sur 7
* Orc = 1 chance sur 8
* Gobelin = 1 chance sur 3

Les PV max :

* Humain = 40
* Elfe = 50
* Orc = 60
* Gobelin = 30

Les PE max :

* Humain = 200
* Elfe = 250
* Orc = 160
* Gobelin = 230

L’XP de départ :

* Humain = 4
* Elfe = 6
* Orc = 9
* Gobelin = 3

### Carte

La carte est de taille fixe (50x50) avec des obstacles à des positions aléatoires.

Les safes zones sont disposées dans chaque coins de la carte et sont de taille fixe : 10x10

La safes zones des Humains est en haut à gauche, des elfes en haut à droite, des gobelins en bas à gauche et des orcs en bas à droite.

Lorsqu’un individu est dans sa safe zone, il regagne 5 PE par tour.

Dans une safe zone, les combats sont interdits.

### Déplacements

Pour les déplacements, lors du début de son tour, l’individu interroge les cases autour de lui, il sélectionne ensuite une case libre (sans obstacle) et se déplace dans la direction de cette case.

A chaque fois qu’il avance d’une case, il questionne la case suivante dans cette même direction, si la case est libre, il continu, s’il y a un obstacle, il s’arrête.

Un obstacle peut-être un obstacle disposé sur la carte mais aussi un allié ou un ennemi.

Dans le cas où « l’obstacle » est un allié ou un ennemi cela entraîne une rencontre.

A chaque fois qu’un individu se déplace d’une case, il perd un PE. Lorsqu’il atteint moins de 25% de ses PE max, il se dirige obligatoirement vers sa safe zone. Si ses PE atteignent 0, il s’arrête sur la case sur laquelle il se trouve. Si un combat est engagé et qu’un des individus à ses PE égal à 0, cet individu perd le combat immédiatement.

### Rencontres

Les rencontrent entre deux individus :

* D’une même alliance (alliés) mais de population différente entrainent un gain d’XP pour les deux individus.
* D’une même population entrainent un gain de PV pour les deux individus, si les PE d’un des deux individus est égal à 0, les PE des deux individus sont partagés entre eux.
* D’alliances différentes (ennemis) entrainent un combat

### Combat

Quand un combat s’engage, l’attaquant frappe le premier, il inflige un nombre de dégât proportionnel à ses XP. Plus il a d’XP plus il inflige de dégâts. L’individu frappé perd un nombre de PV égal au nombre de dégâts infligé.

C’est ensuite à l’autre individu d’attaquer.

Les individus continuent d’attaquer les uns après les autres jusqu’à ce que les PV d’un des individus tombent à 0.

A ce moment, l’individu ayant perdu le combat est retiré de la partie et l’individu ayant gagné prend sa place sur la carte et récupère tous ses XP.

## Déroulement d’une partie

Le déroulement d’une partie est axé autour de la classe « GameManager ».

Au lancement du programme, la carte est créée et affichée et la partie est lancée.

Un tri aléatoire est effectué pour définir l’ordre d’action de chaque équipe, une fois cet ordre établi il restera le même pour toute la partie.

La 1e équipe démarre, tous ses individus se déplacent chacun leur tour et interagissent en fonction d’éventuelles rencontres, combats, niveau de PE etc.

Une fois que le dernier individu de l’équipe 1 s’est déplacé, c’est au tour de l’équipe 2 et ainsi de suite jusqu’à ce que toutes les équipes aient jouées. Une fois que la dernière équipe a joué, le tour se termine et un nouveau tour démarre en faisant jouer les équipes dans le même ordre.

A la fin du tour de chaque personnage une vérification est effectuée sur le nombre d’individus de chaque alliance, si une alliance (gentils ou méchants) tombe à 0 individus, cette alliance est déclarée perdante et l’autre alliance remporte la partie. La partie s’arrête alors.

## Description du code

Dans cette partie nous décrieront le code et comment nous avons organisé notre projet pour répondre aux attentes des consignes.

### Les 3 concepts fondamentaux

#### Héritage

Un héritage est présent sur la classe « Entity » (classe mère) qui donne 2 classes filles : « Gentil » et « Méchant ».

Ces classes Gentil et Méchant sont elles mêmes les classes mères des classes « Humain » et « Elfe » (classes filles de la classe « Gentil ») et « Orc » et « Gobelin » (classes filles de la classe « Méchant »)

Cet héritage est bien visible sur le diagramme de classe présent plus tard dans le document.

#### Encapsulation

Il y a de nombreux exemple d’encapsulation dans ce projet, un des exemples les plus parlant est présent dans la classe « Case » où les attributs x, y, caseType et entity sont des variables private auxquelles on ne peut accéder qu’en utilisant des getters et des setters.

#### Polymorphisme

Exemple de polymorphisme dans notre programme : La méthode attack() qui est définie de manière abstraite dans la classe Entity est un exemple de polymorphisme à travers ses classes filles (Humain, Elfe, Orc et Gobelin). En effet, chacune d’elle implémente de manière différente la méthode attack()

### Singleton

La classe WorldMap est un singleton.

Au lancement du programme, une instance est créée dans cette classe. Cette classe ne peut, par la suite, pas être réinstanciée.

### Variables statiques

La plupart des variables statiques sont réunies dans la classe Rules.

Elles servent à définir les valeurs de PE, PV et XP max ainsi que la taille de la carte.

### Nombre pseudo-aléatoire

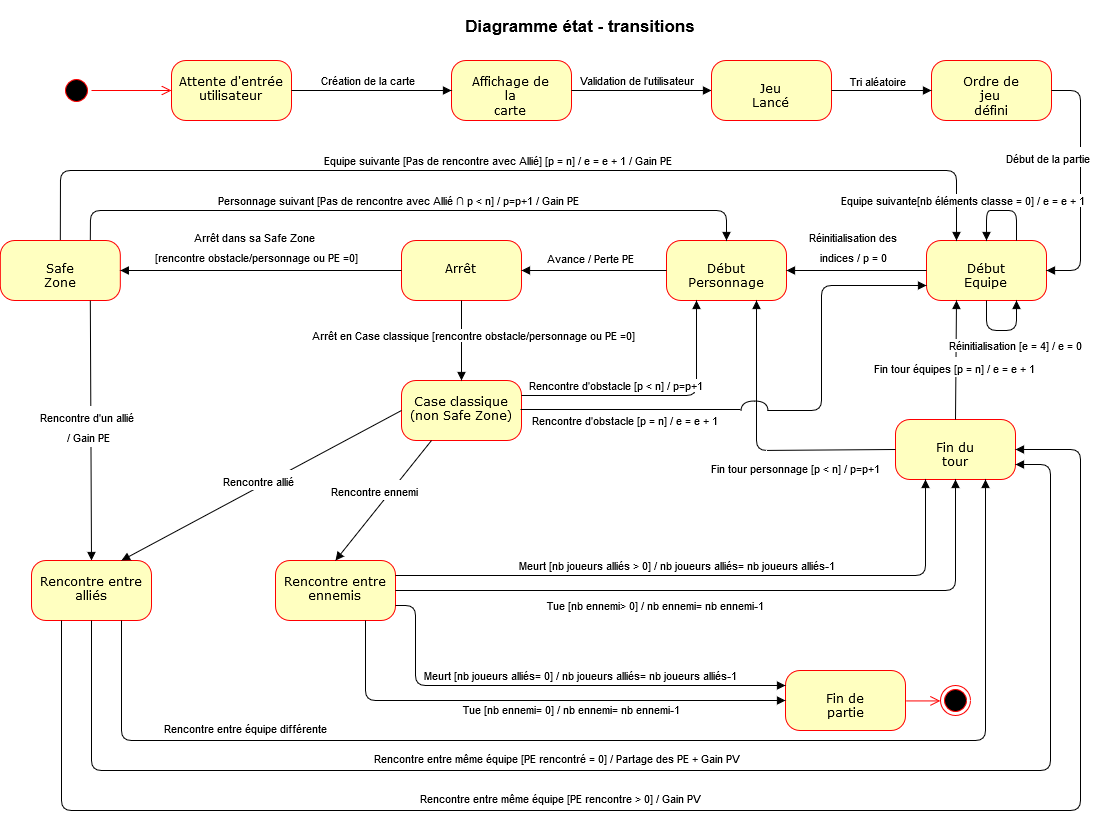
Exemple d’utilisation d’un nombre pseudo aléatoire dans notre programme : Dans la classe WorldMap, la méthode positionnementEntity() utilise un nombre pseudo-aléatoire

Lors de la création de la partie, un nombre pseudo-aléatoire est utilisé afin de définir la probabilité d’apparition d’un individu sur une case safe zone de sa population. Cela permet de faire apparaitre les individus de chaque population sur des cases aléatoire de leur safe zone respective au début de chaque partie. Le nombre d’individus par équipe est donc aussi pseudo-aléatoire.

# Diagrammes UML

## Diagramme de classes

## Diagramme d’états-transitions



Ce diagramme d’états-transitions présente le déroulement d’une partie du début à la fin.

Les états présents de « Attentes d’entrée utilisateur » à « Ordre de jeu défini » sont gérés par la classe GameManager.

A l’état « Début Equipe », la variable e est fixée à 0. Cette variable a pour but de définir quelle équipe va jouer pendant ce tour, e=0 pour la première équipe, e=1 pour la 2e etc.

La valeur de cette variable augmente de 1 à la fin du tour de chaque équipe.

Lorsque la valeur atteint 4, la valeur de la variable est réinitialisée et repasse à 0, démarrant un nouveau tour avec l’équipe 1.

Si une équipe a un nombre d’individus égal à 0 (équipe retirée du jeu), la valeur de e augmente de 1 pour passer à l’équipe suivante.

Lors de la transition « Début Equipe » 🡪 « Début Personnage », on réinitialise la valeur de la variable p à 0. Cette variable permet de savoir combien d’individus ont joués dans l’équipe dont c’est le tour de jour.

A la fin du tour de chaque individu, tant que p<n (n étant le nombre d’individu total dans l’équipe), la valeur de p est augmentée de 1 et c’est à l’individu suivant de l’équipe de jouer.

Lorsqu’à la fin du tour d’un individu, p=n, tous les individus de cette équipe ont joués. C’est donc à l’équipe suivante de jouer.

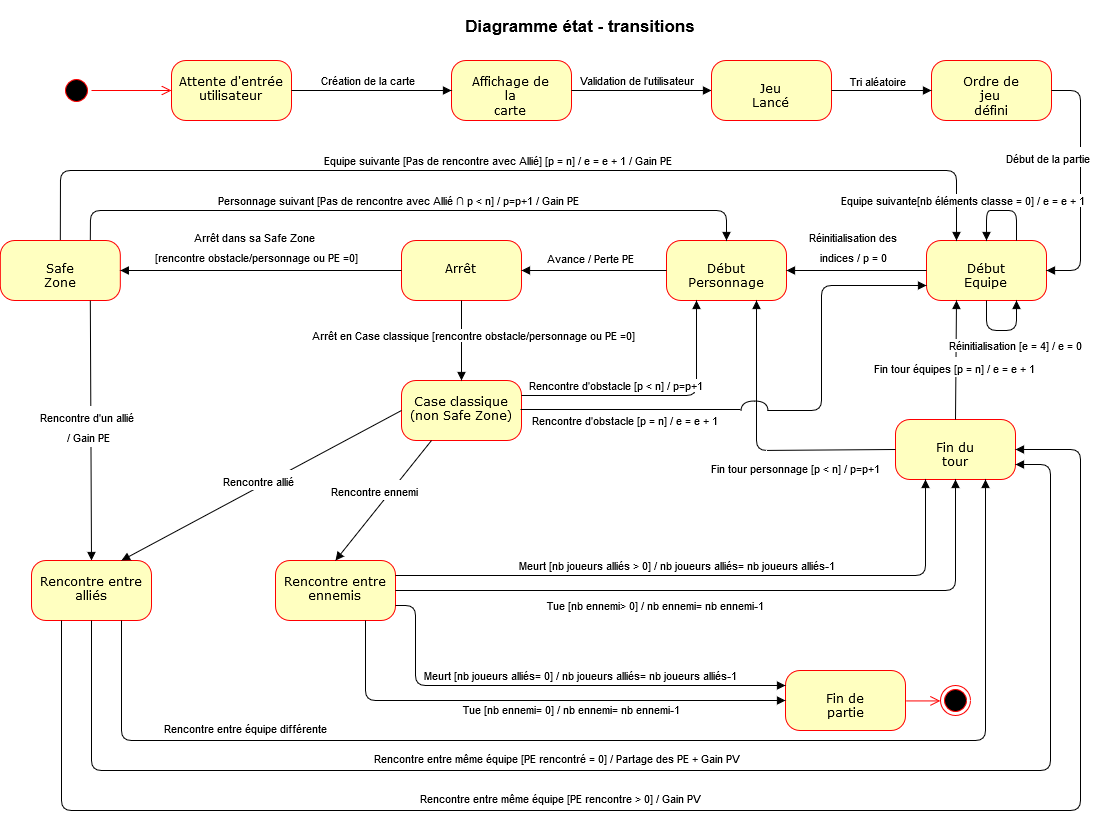
L’état « Arrêt » correspond à l’arrêt d’un individu et entraine de multiples options en fonction de la nature de la case sur laquelle l’individu s’arrête :

* Si c’est dans une safe zone, l’individu récupère les bonus liés à la safe zone et on passe au personnage suivant.
* Si l’arrêt se fait dans une case classique, c’est à l’individu suivant de jouer
* Si l’arrêt entraine une rencontre il y a alors 2 possibilités, soit une rencontre entre alliés soit entre ennemis.

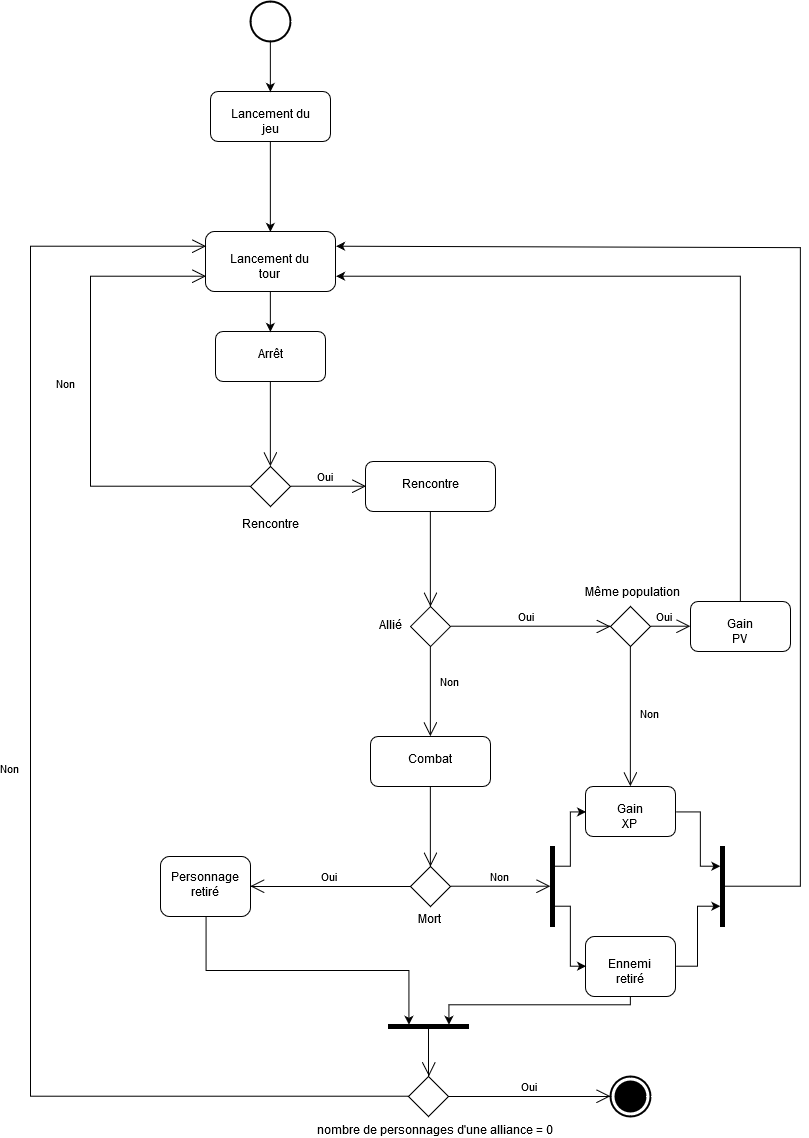
Dans le cas d’une rencontre entre alliés, là encore il y a deux possibilités, si les alliés sont de populations différentes ou de même population. Les bonus récupérés sont alors différents mais la finalité reste la même, la fin du tour de ce personnage et le début du tour du personnage suivant.

Dans le cas d’une rencontre entre ennemis, un combat s’engage, il se déroule comme décrit dans la partie « Nos Choix ». Un combat se solde toujours par la mort d’un des deux individus engagés dans le combat. Une fois le combat terminé, on diminue la valeur de n de 1 dans l’équipe de l’individu ayant perdu le combat.

Si une alliance tombe à 0 individus, la partie se termine, sinon c’est à l’individu suivant de jouer.



## Diagramme d’activités



Ce diagramme représente les différentes activités de la simulation.

Une partie démarre par le lancement du tour, ensuite chaque tour se déroule comme décrit plus en détail dans le diagramme d’états-transitions.

Ce diagramme est surtout utile pour avoir une vision globale de la simulation et des différentes possibilités à chaque état, notamment à partir d’un arrêt qui peut entrainer une rencontre et toutes les options qui en découlent (gain de PV, combat, partage des PE etc.)

# Gestion du projet

Pour la gestion de projet, nous avons désigné Zoé comme notre chef de projet car elle a une bonne vision d’ensemble du projet et aussi car elle avait la volonté de mettre en application les techniques de gestion de projet apprises pendant son sage et aussi en vue de les appliquer pendant son contrat professionnel de cette année.

Nous avons aussi divisé le groupe en deux équipes :

* Jules et Clément qui se sont concentrés sur la structure du projet et la création des différents diagrammes.
* Zoé et Benjamin qui ont plus travaillés sur le code lui-même.

Nous nous rencontrions régulièrement pour mettre en commun l’avancement de chaque équipe afin que tout le monde puisse avoir une vision globale du projet et prendre part au travail de chaque équipe.

Pour mettre en commun notre travail nous avons créé un github que nous avons géré avec gitkraken ce qui nous a aidé à travailler en parallèle sur le code sans risquer d’écraser le travail des autres membres du groupe et aussi à accéder à leur travail facilement.

# Conclusion

Dans ce projet nous avons pu mettre en application tout ce que nous avons pu voir dans le cours, comme les concepts fondamentaux de la programmation java objet comme l’héritage, l’encapsulation et le polymorphisme.

Nous avons aussi pu créer des diagrammes UML dans un contexte un peu plus pratique que les diagrammes réalisés en cours, cela nous a permis de nous rendre compte de l’utilité de ces diagrammes pour l’organisation et la structure de notre projet (notamment grâce au diagramme de classe et d’états-transitions) mais aussi pour avoir un aperçu plus global de notre projet (avec par exemple le diagramme d’activité).