

Nom : Florian VICTORIA DIT THOME

Thomas SCHAAL

Branche: *ISI*

Résumé :

**Introduction**

**I – Diagramme de Cas d’Utilisation**

**II – Diagramme de Classes**

**III – Diagramme de Séquence**

**Conclusion**

UV : *LO02*

Semestre : *Automne 2014*

Modélisation UML:

*La bataille norvégienne*

# Introduction

Afin de mettre en pratique ce qui a été appris durant le cours de « Principe et pratique de la programmation objets » un projet est proposé. Ce projet consiste à l’étude et au développement d’un jeu de bataille norvégienne. Les technologies utilisées dans ce projet sont le langage JAVA et la modélisation UML. Nous effectuons ce projet en binôme, formé par Florian VICTORIA DIT THOME et Thomas SCHAAL. Dans l’optique de mener à bien ce projet, il a d’abord fallu comprendre le fonctionnement du jeu de la bataille norvégienne. Cette partie va donc être détaillée ci-dessous.

Le début de la partie :

La partie est jouable par 2 à 11 joueurs et commence par la distribution des cartes. Chaque joueur reçoit 9 cartes. Les trois premières, il les pose devant lui en ligne face cachée. Les trois suivantes sont à poser sur les trois précédentes mais face visible par tous. Les trois dernières sont à prendre en main à l’abri des regards des autres joueurs. Les cartes non distribuées vont servir de pioches pour le jeu, il faut donc les poser sur le tapis.

Chaque joueur peut échanger les cartes qu’il a en main avec celle face visible qui sont sur le tapis. Cette règle est valable uniquement avant le début d’une partie.

Ensuite, le jeu commence par la personne qui est à gauche du joueur qui a distribué. Les joueurs posent une ou plusieurs carte(s) sur le tapis un après l’autre. Chaque joueur ne peut que poser une carte dont la valeur est supérieur à celle sur le tapis (la couleur ou le signe ne compte pas pour ce jeu). Le joueur peut poser plusieurs cartes de la même valeur en même temps. Quand le joueur a posé une ou plusieurs cartes, il doit piocher des cartes pour maintenir 3 cartes dans sa main. Quand un joueur ne peut plus poser de carte d’une valeur supérieure la dernière carte visible jouée sur le tapis, alors ce joueur ramasse le paquet (bêche) et le met dans sa main.

Toutes les cartes ne sont pas équivalentes. En effet, il existe des cartes spéciales, qui ont un effet bien précis, et il faut les utiliser de manière stratégique pour remporter la victoire et être « Danish ».

* Le 2 est une carte spéciale qui est supérieure aux autres, quand on l’a joué, le jeu repart à la valeur 2.
* Le 7 est une carte spéciale qui oblige le joueur suivant à jouer une carte d’une valeur inférieure ou égale à 7.
* Le 8 est une carte spéciale qui oblige le joueur suivant à passer son tour, le joueur suivant peut poser uniquement un 8, ce qui obligera le joueur suivant à passer son tour ou lui-même poser un autre 8. Et ainsi de suite.
* Le 10 est une carte spéciale qui banni les cartes présentes sont le tapis de manière définitive.
* L’As est une carte spéciale qui permet à celui qui pose la carte de choisir le joueur de son choix et lui donner le paquet de jeu.

Ce livrable comporte une analyse du diagramme des cas d’utilisation suivie par l’étude du diagramme de classes et du diagramme de séquence. Enfin nous conclurons en nous interrogeant sur les aspects sûrs de la modélisation et ceux qui seront voués à évoluer.

Table des matières

[Introduction 2](#_Toc403034568)

[I. Diagramme de cas d’utilisation 4](#_Toc403034569)

[II. Diagramme de classes 4](#_Toc403034570)

[III. Diagramme de séquence 4](#_Toc403034571)

[Conclusion 4](#_Toc403034572)

# I. Diagramme de cas d’utilisation

# II. Diagramme de classes

# III. Diagramme de séquence

# Conclusion

Pour le moment, les trois diagrammes sont faits et nous savons qu’une partie sera correcte par rapport à ce qui va être implémenté. Cependant, pour l’autre partie on est moins sûre. Nous pensons que pour la première partie par exemple, le nombre de classes est sûrement définitif ainsi que les zones de jeu. Mais à l’inverse, pour la deuxième partie, nous pensons qu’il y aura des changements dans le nombre d’attributs et le nombre de méthodes de chaque classe. L’encapsulation de certain attributs et méthodes seront aussi à changer, certains vont devoir passer en «protected » pour les besoins de l’application. Dans tous les cas, nous sommes prêts à nous adapter aux changements entre la conception et l’implémentation.