AVANZINO Aurélien – GOURDON Stéphanie

Documentations complète sur le projet d’implémentation du dilemme du prisonnier à réaliser par groupe de deux

Patrons et Composants

Spécifications – Dilemme du prisonnier

**HISTORIQUE DES MODIFICATIONS**

|  |  |  |
| --- | --- | --- |
| AUTEUR | LIBELLE | DATE MODIFICATION |
| GOURDON Stéphanie | Rédaction initiale du document |  |
| AVANZINO Aurélien | Ajout des spécifications version 2.0 |  |
| GOURDON Stéphanie | Ajout des spécifications version 2.5 |  |
| GOURDON Stéphanie | Suppression des clones Strategies |  |
| AVANZINO Aurélien | Ajout des design patterns respectés |  |
| AVANZINO Aurélien | Ajout des spécifications version 3.0 |  |

***INFORMATIONS COMPLEMENTAIRES :***

* Dépôt Gitlab du projet : <https://gitlab.com/AurelienAVZN/pc_dilemmeduprisonnier>
* JavaDoc du projet : <https://pc-dilemmeprisonnier.netlify.app/> (disponible également dans le dossier javadoc du projet)
* Binôme partenaires rendu N°4 : Mariia Selivanova – Marie-Josée Bassil
* Adresses Mail :
  + AVANZINO Aurélien : [aurelien.avanzino@etu.univ-grenoble-alpes.fr](mailto:aurelien.avanzino@etu.univ-grenoble-alpes.fr)
  + GOURDON Stéphanie : [stephanie.gourdon@etu.univ-grenoble-alpes.fr](mailto:stephanie.gourdon@etu.univ-grenoble-alpes.fr)

Table des matières

[I. Introduction 3](#_Toc60639810)

[II. Le Dilemme du Prisonnier 3](#_Toc60639811)

[II.1 Description du projet 3](#_Toc60639812)

[II.1 Introduction 3](#_Toc60639813)

[II.2 Présentation du dilemme 3](#_Toc60639814)

[II.3 Une version implémentable du dilemme 3](#_Toc60639815)

[II.2 Spécifications 3](#_Toc60639816)

[II.2.1 L’application à réaliser 3](#_Toc60639817)

[II.2.2 Exemples de stratégie 3](#_Toc60639818)

[II.2.3 Interaction entre les stratégies 4](#_Toc60639819)

[III. Comportement du logiciel 4](#_Toc60639820)

[III.1 Version 2.6 4](#_Toc60639821)

[III.1.1 Diagramme de classe 4](#_Toc60639822)

[III.1.2 Diagramme de séquence 4](#_Toc60639823)

[III.2 Version 3.2 4](#_Toc60639824)

[III.2.1 Diagramme de classe 4](#_Toc60639825)

[III.2.2 Diagramme de séquence 5](#_Toc60639826)

[V.2 Diagramme de séquence 6](#_Toc60639827)

[IV. Développements et Maintenances correctives, évolutives 6](#_Toc60639828)

[VI.1 Mise en place des design patterns GRASP et Strategy 6](#_Toc60639829)

[VI.1.1 Implémentation actuelle 7](#_Toc60639830)

[VI.1.2 Objectifs et but 7](#_Toc60639831)

[VI.1.3 Développement et modifications 7](#_Toc60639832)

[VI.2 Séparer l’IHM du programme principal 7](#_Toc60639833)

[VI.2.1 Implémentation actuelle 7](#_Toc60639834)

[VI.2.2 Objectifs et but 7](#_Toc60639835)

[VI.3 Suppression de la fonction clone() de l’interface Stratégie 7](#_Toc60639836)

[VI.3.1 Implémentation actuelle 7](#_Toc60639837)

[VI.3.2 Objectifs et but 7](#_Toc60639838)

[VI.3.3 Nouvelle implémentation 8](#_Toc60639839)

[VI.3.4 Ce que nous ferons plus tard 8](#_Toc60639840)

[VI.4 Développement d’une interface graphique 8](#_Toc60639841)

[VI.4.1 Objectifs et but 8](#_Toc60639842)

[VI.4.2 Implémentation actuelle 8](#_Toc60639843)

# Introduction

L’objectif du projet consiste à proposer un modèle UML d’une petite application permettant de mettre en œuvre des tournois de stratégies jouant au *Dilemme du prisonnier*, puis de proposer une implémentation en JAVA de cette application.

Des modifications et/ou fonctionnalités peuvent être ajoutées au fur et à mesure suivant les nouvelles spécifications demandées par le client.

# Le Dilemme du Prisonnier

## II.1 Description du projet

### II.1 Introduction

Le *Dilemme du prisonnier* est un modèle très simple de la théorie des jeux. Il semble appréhender en miniature les tensions entre cupidité individuelle et intérêts de la coopération collective. Pour cette raison, il est devenu un des modèles les plus utilisés en sociologie, biologie et économie.

### II.2 Présentation du dilemme

Deux suspects porteurs d’armes ont été arrêtés devant une banque et mis dans deux cellules séparées. Les deux prévenus ne peuvent pas communiquer et doivent choisir entre avouer qu’ils s’apprêtaient à commettre un hold-up ou ne rien avouer. Les règles que le juge leur impose sont les suivantes :

* Si l’un avoue et pas l’autre, celui qui avoue sera libéré en remerciement de sa collaboration et l’autre sera condamné à cinq ans de prison ;
* Si aucun n’avoue, ils ne seront condamnés qu’à deux ans de prison, pour port d’arme illégal ;
* Et si les deux avouent, ils iront faire chacun quatre ans de prison.

Dans cette situation, il est clair que si les deux s’entendent (pas d’aveu), ils s’en tireront globalement mieux (2 x 3 ans de remise de peine) que si l’un des deux dénonce l’autre (1 x 5 ans de remise de peine). Mais alors l’un peut être tenté de s’en tirer encore mieux en dénonçant son complice. Craignant cela, l’autre risque aussi de dénoncer son complice pour ne pas être le dindon de la farce. Le dilemme est donc : Faut-il accepter de couvrir son complice (donc coopérer avec lui) ou le trahir ?

### II.3 Une version implémentable du dilemme

Ce dilemme peut devenir beaucoup plus intéressant et plus réaliste lorsque la durée de l’interaction n’est pas connue. On peut envisager de se souvenir du comportement d’un joueur à son égard et développer une stratégie en rapport. Par exemple, si je sais que mon adversaire ne coopère jamais, mon intérêt sera de ne pas coopérer non plus, sous peine d’être systématiquement floué. Par contre si je sais que mon adversaire coopèrera toujours quoi qu’il arrive, j’aurai intérêt à être vicieux et ne jamais coopérer pour maximiser mon gain.

## II.2 Spécifications

### II.2.1 L’application à réaliser

L’application à réaliser doit permettre de mettre en œuvre des tournois de stratégies jouant au dilemme des prisonniers tel que décrit dans la section II.1.3. Les stratégies sont implémentées en JAVA. Il faudra implémenter les stratégies décrites en section (insérer la section) plus vos stratégies personnelles ou celles de vos collègues. Un tournoi est la confrontation d’un ensemble de stratégies. L’ensemble des stratégies est un sous-ensemble, choisi par l’utilisateur, des stratégies disponibles. Une rencontre entre deux stratégies se joue en n tours, n étant également défini par l’utilisateur. Dans un tournoi, une stratégie doit rencontrer toutes les stratégies, elle-même comprise, de l’ensemble sélectionné. Le score réalisé par une stratégie est la somme de ses points récoltés lors de chacune des confrontations.

### II.2.2 Exemples de stratégie

* Gentille : Je coopère toujours
* Méchante : Je trahis toujours
* Donnant-Donnant : Je coopère à la première partie, puis je joue ce qu’a joué l’autre à la partie précédente.
* Donnant-Donnant-Dur : Je coopère à la première partie, puis je coopère sauf si mon adversaire a trahi lors de l’une des deux parties précédentes
* Méfiante : Je trahis à la première partie, puis je joue ce qu’a joué l’autre à la partie précédente
* Rancunière : Je coopère à la première partie, mais dès que mon adversaire trahit, je trahis toujours.

### II.2.3 Interaction entre les stratégies

On suppose que deux stratégies lors d’une confrontation ne peuvent pas passer d’accord. La seule information qu’une tribu connaît sur l’autre est son comportement passé lors des coups précédents. Les décisions des deux tribus lors de la partie sont prises simultanément. Les nombre de parties n’est pas connu à l’avance.  
Par rapport au simple dilemme des prisonniers nous choisissons de rajouter la possibilité de renoncer à jouer, mais ce refus est définitif.

De manière plus abstraite, deux entités peuvent choisir entre coopérer (notation c), trahir (notation t) ou renoncer (notation n). Si l’une trahit et l’autre coopère (partie [t, c]), celle qui trahit obtient un gain de T unités et celle qui coopère (et donc s’est fait duper) obtient un gain de D unités. Lorsque les deux entités coopèrent (partie [c,c]), elles gagnent chacune C unités en récompense de leur association. Quand elles trahissent toutes les deux (partie [t, t]), elles gagnent P unités pour s’être laissé piéger mutuellement. Si une partie n’as pas lieu parce qu’une l’une a refusé de jouer les deux entités gagnent N unités. Le choix des coefficients T, D, C, P et N n’est pas fortuit. Conformément aux n°181 de POUR LA SCIENCE nous prenons : T = 5, D = 0, C = 3, P = 1, N = 2.

# Comportement du logiciel

Les diagrammes de classes et de séquences représentent l’implémentation des dernières versions. Actuellement ils existent deux versions développées en parallèle. La version 2.6 qui possède un affichage console et la version 3.2 possédant un affichage graphique (développé via javax.swing), cependant dans leur implémentation et fonctionnement ces dernières sont identiques.

## III.1 Version 2.6

### III.1.1 Diagramme de classe

**PARTIE FRONT DE L’APPLICATION**

**PARTIE BACK DE L’APPLICATION**

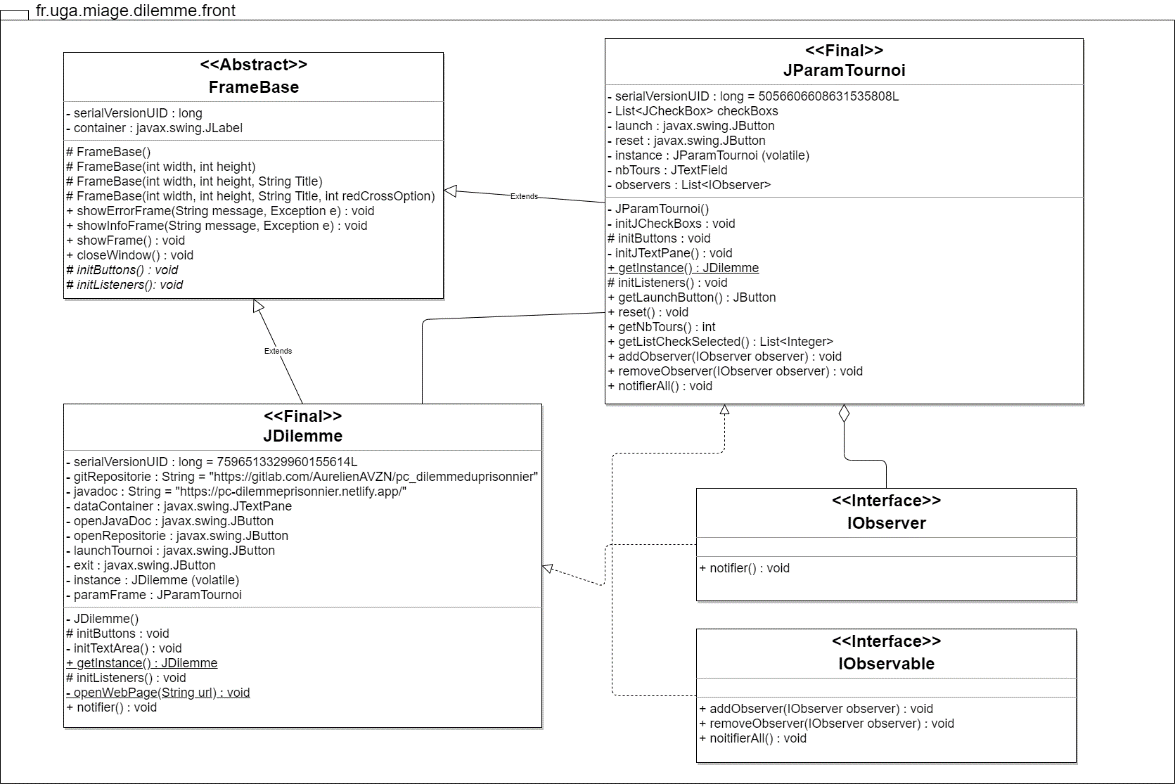
**DIAGRAMME DE CLASSE COMPLET**

### III.1.2 Diagramme de séquence

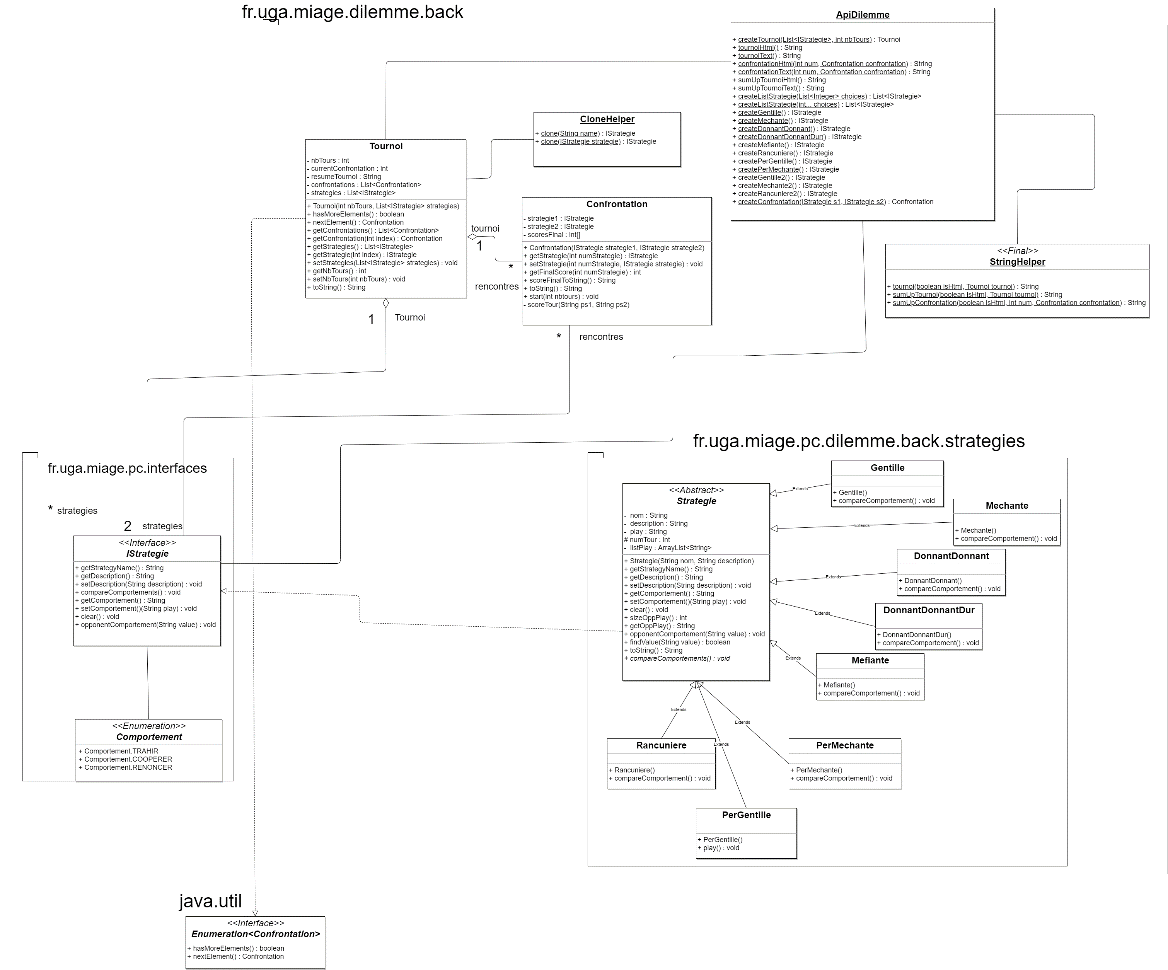
## III.2 Version 3.2

### III.2.1 Diagramme de classe

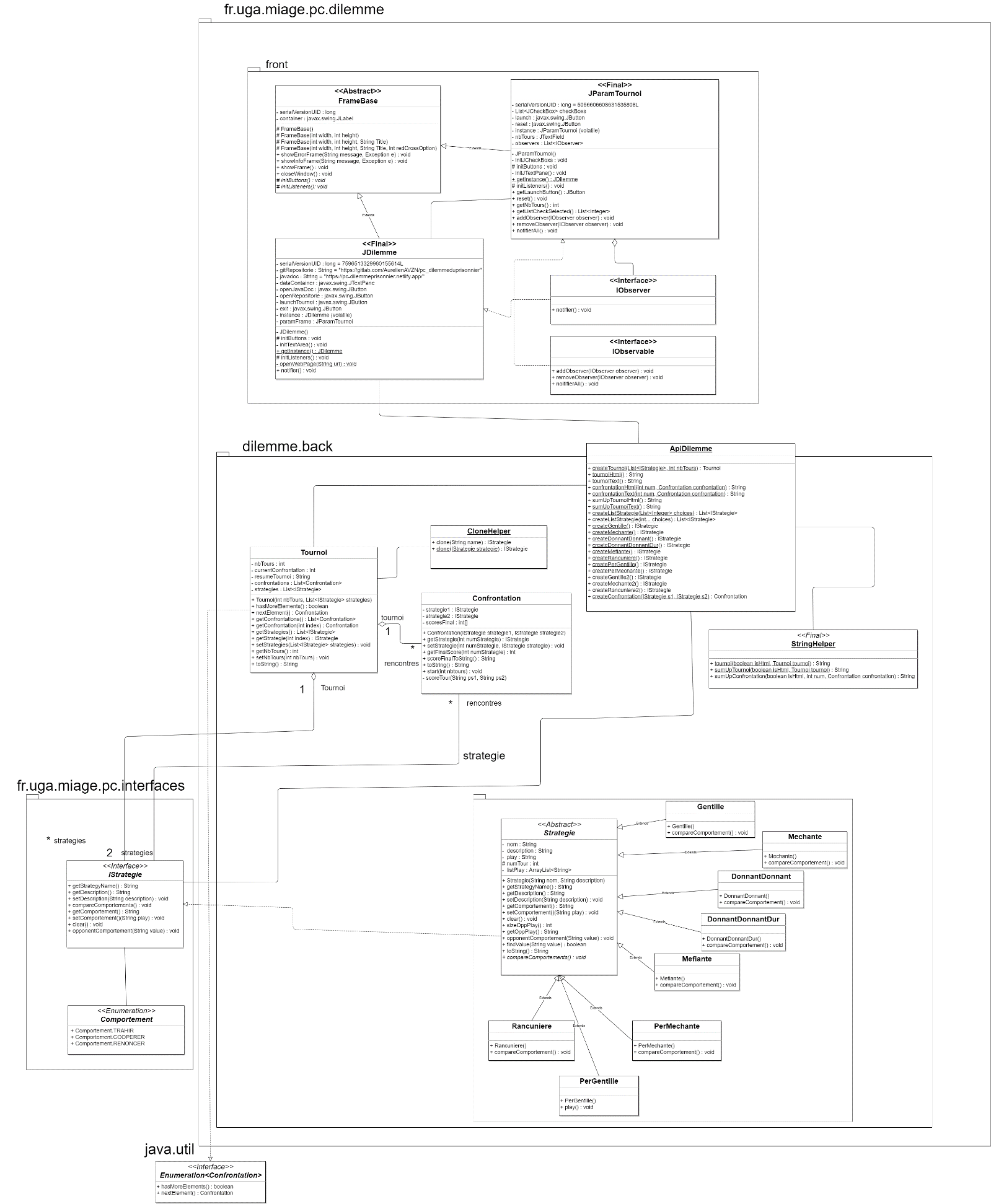
**PARTIE FRONT DE L’APPLICATION**



**PARTIE BACK DE L’APPLICATION**



**DIAGRAMME DE CLASSE COMPLET**



### III.2.2 Diagramme de séquence

# Développements et Maintenances correctives, évolutives

## VI.1 Mise en place des design patterns GRASP et Strategy

Mis en place du développement depuis la version 2.0 du dilemme des prisonniers

### VI.1.1 Implémentation actuelle

Actuellement la disposition des classes utilisées dans l’application telle que décrite dans le diagramme de classe de la version 1.0 (cf. section ????) ne respecte pas particulièrement des types design patterns.   
L’objectif de la première version a été en priorité de fournir une implémentation fonctionnelle respectant l’ensemble des spécifications demandés par l’utilisateur.

Cependant, il existe une prémisse du design pattern Strategy au travers de l’implémentation des stratégies car ces dernières héritent d’une classe mère Strategie.

### VI.1.2 Objectifs et but

Le but de ce développement est de respecter les règles établies pour les design pattern suivant :

* Le design pattern GRASP : Se composent de lignes directrices pour l’attribution de la responsabilité des classes et des objets.
* Le design pattern Strategy : De type comportemental, il est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application. Il permet de définir une famille d’algorithme, encapsuler chacun d’eux en tant qu’objet, et les rendre interchangeables.

### VI.1.3 Développement et modifications

(TODO)

## VI.2 Séparer l’IHM du programme principal

Mise en place de développement depuis la version 2.5 du dilemme des prisonniers

### VI.2.1 Implémentation actuelle

Actuellement plusieurs implémentations dans l’application posent ou peuvent poser problème.  
Dans un premier temps l’implémentation de l’IHM se trouve directement dans le Main de l’application.  
De plus la partie front communique directement avec la partie back de l’application alors qu’il faudrait qu’il est une classe intermédiaire pour contrôler la communication entre les deux (cf. section XXXX)

### VI.2.2 Objectifs et but

L’objectif de ce développement est de régler les différents points à améliorer. Pour cela nous allons réaliser les développements et modification suivantes :

* Création d’une classe intermédiaire qui se chargera d’effectuer la liaison entre la partie front ainsi que la partie back pour la communication.
* Séparation des méthodes front présent dans le Main dans une classe à part afin toutes les parties dissociés (Back, Front, et Main)

## VI.3 Suppression de la fonction clone() de l’interface Stratégie

Mise en place du développement depuis la version 2.5 du dilemme des prisonniers

### VI.3.1 Implémentation actuelle

Actuellement afin de respecter la spécification sur les confrontations qui est : « *Une stratégie se confronte aux autres stratégies* ***y compris elle-même*** »  
L’interface IStrategie hérite de l’interface Cloneable fourni avec JAVA permettant ainsi d’avoir la signature de la fonction clone().

En effet cette dernière nous permet, lors de l’initialisation du Tournoi et des Confrontations de créer une copie de la stratégie afin de pouvoir dissocier les scores de chacune mais aussi les coups joués par l’adversaire qui sont enregistrées

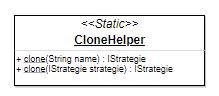
### VI.3.2 Objectifs et but

L’objectif de la suppression des clones est de rendre l’interface IStrategie commune avec l’ensemble des implémentations effectuées par le second binôme avec lequel nous travaillons. (cf. Informations complémentaires)  
Leur implémentation de la spécification citée ci-dessus (cf. section VI.3.1) n’utilise pas de méthode clone ou équivalent. Ce dernier passe par une classe intermédiaire qui enregistre le score.

Cependant, il nous est impossible de supprimer le fonctionnement de la méthode clone() sans modifier le fonctionnement de la classe Tournoi. Il faut donc dans un premier temps trouver une nouvelle implémentation afin de palier la suppression de la méthode clone dans l’interface sans modifier la classe Tournoi.

### VI.3.3 Nouvelle implémentation

Afin de palier à cette suppression, il a été décidé d’ajouter une nouvelle classe static qui se chargera d’effectuer le clone de la stratégie. Cette classe nommé CloneHelper, propose deux méthodes clone, une prenant le nom de la stratégie à cloner et l’autre prend directement la IStrategie.



### VI.3.4 Ce que nous ferons plus tard

Il faudra peut-être envisager une implémentation différentes pour les fonctions clone(), car ces dernières fait des tests sur le type d’instance de l’objet, ce qui empêche la généricité de l’application.  
En effet, si de nouvelles stratégies sont à implémenter il faudra alors modifier l’implémentation de ces stratégies pour les prendre en compte.

## VI.4 Développement d’une interface graphique

Mise en place du développement depuis la version 3.0 du dilemme des prisonniers

### VI.4.1 Objectifs et but

### VI.4.2 Implémentation actuelle