



UNIVERSITÉ PIERRE ET MARIE CURIE

A.N.D.R.O.I.D.E.

PROJET DU SECOND SEMESTRE DE MASTER 1

---

**PANDROIDE Mafia de Cuba**  
RAPPORT

---

*Auteurs :*

Pierre-François MONVILLE  
Gilles NOUVEAU  
Beatriz ROJAS  
Claire SALOMÉ

*Encadrants :*

Bénédicte LEGASTELOIS  
Nicolas MAUDET

## Sommaire

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>I</b>	<b>Cahier des charges</b>	<b>3</b>
<b>2</b>	<b>Présentation du projet</b>	<b>3</b>
2.1	Contexte . . . . .	3
2.2	Objectifs du projet . . . . .	3
2.3	Présentation du jeu Mafia de Cuba . . . . .	3
<b>3</b>	<b>Expression fonctionnelle du besoin</b>	<b>4</b>
3.1	Fonctions de service de base . . . . .	4
3.2	Fonctions de service optionnelles . . . . .	5
3.3	Fonctions de contrainte . . . . .	5
<b>4</b>	<b>Cadre de réponse</b>	<b>5</b>
4.1	Fonctionnalités de base : Description de l'interface . . . . .	5
4.1.1	Écran d'accueil . . . . .	5
4.1.2	Lire les règles du jeu . . . . .	5
4.1.3	A propos . . . . .	5
4.1.4	Démarrer une nouvelle partie . . . . .	5
4.1.5	Écran de la partie . . . . .	6
4.2	Fonctionnalités optionnelles . . . . .	6
4.2.1	Paramètres de l'interface . . . . .	6
4.2.2	Niveau d'honnêteté des IA . . . . .	6
4.2.3	IA du Parrain . . . . .	7
4.2.4	Différents types de stratégie pour chaque rôle . . . . .	7
4.2.5	Ajouter ses propres stratégies . . . . .	7
4.2.6	IA dynamique . . . . .	7
4.2.7	Visualisation des connaissances . . . . .	7
4.3	Éléments de validation du projet . . . . .	8
<b>5</b>	<b>Ressources</b>	<b>8</b>
5.1	Ressources humaines . . . . .	8
5.2	Ressources techniques . . . . .	8
5.3	Planification du projet . . . . .	9
<b>II</b>	<b>Analyse et conception</b>	<b>10</b>
<b>6</b>	<b>Logique de développement</b>	<b>10</b>
6.1	Diagramme de classes . . . . .	11
6.2	Diagramme d'états . . . . .	12

---

<b>7 Fonctionnement des IA</b>	<b>15</b>
7.1 Modélisation des connaissances des joueurs . . . . .	15
7.1.1 Calcul des bornes sur le nombre de mondes avant et après . . . . .	15
7.1.2 Génération des mondes avant et après . . . . .	18
7.1.3 Mise à jour et poids des mondes . . . . .	20
7.2 Stratégie du choix du rôle . . . . .	20
7.2.1 FirstPositionStrategy . . . . .	21
7.2.2 SecondPositionStrategy . . . . .	22
7.2.3 MiddlePositionStrategy . . . . .	23
7.2.4 LastPositionStrategy . . . . .	23
7.3 Stratégie du rôle public à montrer . . . . .	23
7.4 Interactions entre l'utilisateur humain et les IA . . . . .	24
7.5 Génération des réponses des IA . . . . .	25
7.6 Les Stratégies de rôle . . . . .	28
7.6.1 Stratégie du voleur . . . . .	28
7.6.2 Stratégie du fidèle . . . . .	30
7.6.3 Stratégie du nettoyeur . . . . .	30
7.6.4 Stratégie de l'agent . . . . .	30
7.6.5 Stratégie de l'enfant des rues . . . . .	31
7.6.6 Stratégie du chauffeur . . . . .	31
<b>8 Améliorations</b>	<b>31</b>
<b>III Manuel d'utilisation</b>	<b>33</b>
<b>9 Bibliographie</b>	<b>42</b>

## 1 Introduction

Ce document est le rapport du projet "Mafia de Cuba". Il s'inscrit dans le cadre de l'Unité d'Enseignement (UE) "Projet" de la première année de Master Informatique de la spécialité Agents Distribués, Robotique, Recherche Opérationnelle, Interaction, Décision (ANDROIDE) de L'Université Pierre et Marie Curie (UPMC).

Le rapport est divisé en 3 grandes parties :

- Le cahier des charges.
- La description de l'analyse et de la conception du projet.
- Le manuel d'utilisation de l'application développée.

## Première partie

### Cahier des charges

Cette partie du rapport définit le cahier des charges du projet. Elle a pour objectif dans un premier temps de définir le contexte du projet et la problématique à résoudre. Elle exprime les besoins du client, au travers de fonctions de service et de fonctions de contrainte tout en précisant les objectifs principaux et optionnels. De plus, Elle fournit une description du fonctionnement de la solution. Elle définit ensuite les éléments de validation du projet. Et elle présente enfin les ressources humaines, les outils techniques et logiciels et l'organisation dans le temps de la réalisation des objectifs.

## 2 Présentation du projet

### 2.1 Contexte

### 2.2 Objectifs du projet

L'objectif de ce projet est de réaliser un logiciel permettant à un utilisateur humain de configurer et de jouer des parties du jeu de société "Mafia de Cuba" en affrontant plusieurs intelligences artificielles (IA) simulant les autres joueurs de la partie.

Outre la réalisation d'une application jouable, l'aspect recherche de ce projet est porté sur l'étude, la modélisation et l'implémentation de processus cognitifs et de stratégies d'agents dans le cadre d'un jeu de communication avec annonces publiques, non coopératif, à information incomplète et intégrant les notions de mensonge et de bluff.

### 2.3 Présentation du jeu Mafia de Cuba

Mafia de Cuba est un jeu de société, co-créé par Philippe des Pallières et Loïc Lamy, édité chez les éditions "Lui-Même" en 2015. C'est un jeu d'enquête, de communication et de bluff qui se joue de 6 à 12 personnes.

Un joueur incarne le Parrain et confie sa boîte de cigares, qui contient des diamants et des jetons personnage, au premier joueur. La boîte passe ensuite par chaque joueur, qui va en profiter pour soit voler des diamants encore présents dans la boîte, soit prendre un jeton personnage. Après que le Parrain a récupéré la boîte, il interroge les joueurs pour identifier les voleurs et retrouver ses diamants.

Plusieurs rôles sont présents dans la boîte.

Le fidèle : il gagne si le parrain gagne, il a tout intérêt à dire la vérité pour clarifier la vision du monde du parrain.

L'agent : il gagne seulement s'il se fait accuser par le parrain. Il a tout intérêt à faire porter les soupçons sur lui. L'agent s'il se fait accuser gagne seul, même si un autre agent était présent.

Le chauffeur : Il gagne seulement si son voisin de droite (celui qui lui a donné la boîte) gagne la partie. Il doit essayer d'identifier le rôle de son passager et faire en sorte de le faire gagner.

Si un joueur prend un ou plusieurs diamants, il devient voleur et son but est de faire en sorte que le parrain perde en mettant fin à la partie (elle s'arrête si le parrain accuse un fidèle, chauffeur ou enfant des rues et ne dispose plus de joker). Ensuite le voleur qui a dérobé le plus de diamants gagne la partie avec tous les enfants des rues.

Enfin si la boîte est vide ou s'il s'agit du dernier joueur qui décide de ne rien prendre (le seul à avoir cette possibilité) alors le joueur devient enfant des rues et gagne si l'un des voleurs gagne. Son but est donc de faire perdre le parrain.

On distingue ainsi trois camps : le camp du parrain regroupant le parrain et ses fidèles travaillant ensemble pour démasquer l'ensemble des voleurs, le camp des voleurs associé aux enfants des rues, voulant tromper le parrain et enfin le camp des agents où chaque agent essaye de se faire accuser par le parrain. Les chauffeurs sont dans le même camp que leur passager.

Les points essentiels du jeu Mafia de Cuba sont les suivants :

- C'est un jeu non coopératif. Chaque joueur va choisir un rôle en fonction du contenu de la boîte qu'il reçoit. Cela va définir son propre objectif pour remporter la partie.
- C'est un jeu à information incomplète. Au début de la phase d'enquête du Parrain, chaque joueur ne dispose que de peu d'informations sur la configuration de la partie.
- C'est un jeu de communication et de bluff. Dans la seconde phase de jeu, chaque joueur va être amené à échanger avec les autres joueurs des informations sur ce qu'il sait ou pense savoir. Cependant, en fonction de son objectif, ses annonces publiques peuvent être mensongères.

## 3 Expression fonctionnelle du besoin

### 3.1 Fonctions de service de base

- Permettre à l'utilisateur de jouer au jeu de société Mafia de Cuba sur ordinateur.
- Proposer à l'utilisateur de configurer les paramètres de la partie qu'il souhaite jouer.
- Permettre à l'utilisateur de jouer des parties avec d'autres joueurs simulés par des intelligences artificielles.

### 3.2 Fonctions de service optionnelles

- Proposer des paramétrages de l'interface plus avancés.
- Laisser l'utilisateur choisir des configurations modifiant les règles afin d'observer le comportement des stratégies dans un environnement particulier.
- Permettre de définir un niveau d'honnêteté pour les IA.
- Offrir la possibilité à l'utilisateur d'incarner un joueur autour de la table avec les autres IA, dont une simulant le Parrain.
- Proposer différents types de comportements en fonction du rôle choisi par l'IA.
- Autoriser l'utilisateur à importer ses propres stratégies dans le jeu.
- Proposer à l'utilisateur des parties plus vivantes avec des IA dynamiques qui peuvent demander la parole.
- Offrir une visualisation des connaissances d'un agent au cours de la partie.

### 3.3 Fonctions de contrainte

- Respecter les règles du jeu définies par les auteurs.
- Présenter une interface ergonomique pour l'utilisateur humain.
- Implémenter des IA avec un niveau de jeu acceptable.
- Proposer une expérience de jeu agréable pour l'utilisateur avec des IA réactives.

## 4 Cadre de réponse

### 4.1 Fonctionnalités de base : Description de l'interface

#### 4.1.1 Écran d'accueil

Au lancement de l'application, l'utilisateur arrive sur l'écran d'accueil du jeu. A partir de cet écran, il peut démarrer une nouvelle partie, lire les règles du jeu ou en découvrir plus à propos de l'équipe de développement et du contexte du projet.

#### 4.1.2 Lire les règles du jeu

L'utilisateur peut s'informer sur le principe du jeu, le déroulement d'une partie, les objectifs de chacun des personnages et les conditions de victoire.

#### 4.1.3 À propos

L'utilisateur peut en découvrir plus sur l'équipe de développement et du contexte du projet.

#### 4.1.4 Démarrer une nouvelle partie

Cette option ouvre un nouvel écran, où l'utilisateur choisit le nombre de joueurs parmi les nombres autorisés par les règles du jeu. Le contenu de la boîte avec une répartition des rôles par défaut est alors proposé. L'utilisateur, peut ensuite, s'il le souhaite, modifier

ces réglages, tout en respectant le nombre de jetons prévu pour le nombre de joueurs sélectionné. Il peut remplacer un jeton "Fidèle" par le jeton "Nettoyeur" et également modifier le nombre de jokers. L'utilisateur peut enfin démarrer la partie ou revenir au menu d'accueil.

#### 4.1.5 Écran de la partie

Lors de la partie, l'utilisateur incarne le rôle du Parrain. Pour la première phase du vol de diamants, le Parrain indique le nombre de diamants qu'il souhaite écarter du jeu, compris entre 0 et 5. Une animation indique ensuite le début de la phase d'enquête. Lors de cette transition, les IA se seront servies dans la boîte pour choisir leurs rôles.

Au début de la phase d'enquête, le contenu restant de la boîte s'affiche pour le Parrain. Ce dernier peut alors interroger les joueurs dans l'ordre qu'il veut et autant de fois qu'il le souhaite. Il peut également accuser un joueur à tout moment de la partie.

Pour poser une question, l'utilisateur sélectionne d'abord le joueur qu'il souhaite interroger, puis il choisit la question dans la liste des questions disponibles. Les questions sont classées par thème. Une question peut être liée par exemple au contenu de la boîte ou à un rôle. Si le Parrain décide d'accuser un joueur, il sélectionne le joueur et clique sur le bouton d'accusation.

Tout au long de la partie, des aides de jeu sont affichés pour l'utilisateur. Des icônes rappellent le nombre de diamants écartés en début de partie, le nombre de diamants non encore récupérés et le nombre de joker restant. Un historique des questions posées et de leurs réponses est aussi visible. Des icônes cliquables permettent de recommencer une partie ou de revoir les règles du jeu.

## 4.2 Fonctionnalités optionnelles

### 4.2.1 Paramètres de l'interface

Dans le menu d'accueil, l'utilisateur peut accéder aux options lui permettant de modifier la taille de la fenêtre, d'activer ou de désactiver les effets sonores, ou encore de choisir la langue. Le choix des modifications se fait entre plusieurs valeurs fixes.

- Pour la taille de la fenêtre, le choix par défaut est la résolution 1366 x 768. l'utilisateur peut cliquer sur d'autres résolutions proposées, comme par exemple : 1920 x 1020, 1024 x 768, 800 x 600.

- Concernant le son, l'utilisateur peut cliquer sur des boutons ON et OFF.

- Le choix de la langue par défaut est le Français. L'utilisateur peut sélectionner une autre langue parmi celles disponibles, comme l'Anglais ou l'Espagnol par exemple.

### 4.2.2 Niveau d'honnêteté des IA

L'utilisateur peut définir la propension des IA à dire la vérité. Ce niveau d'honnêteté impacte le choix du rôle que va prendre l'IA lors de la phase de vol et la stratégie qu'il va adopter pendant la partie.

#### 4.2.3 IA du Parrain

L'utilisateur peut avoir la possibilité d'incarner un des joueurs autour de la table contre une IA jouant le Parrain.

Dans ce mode de jeu, la personne peut choisir sa place autour de la table ou être positionné aléatoirement. Lors la phase de vol, le contenu de la boîte est présenté au joueur et il lui est demandé de choisir ce qu'il souhaite voler. Pendant la phase d'enquête, lorsque le Parrain lui pose une question, une liste de réponses s'affiche dans laquelle il sélectionne son choix. Il peut aussi demander directement la parole au Parrain pour donner des informations supplémentaires en cours de jeu en utilisant une liste de phrases appropriées.

#### 4.2.4 Différents types de stratégie pour chaque rôle

Dans le menu de configuration de la partie, l'utilisateur peut sélectionner pour chaque type de rôle une stratégie dans une liste de stratégies disponibles proposées par défaut et spécifiques pour le rôle en question.

#### 4.2.5 Ajouter ses propres stratégies

L'utilisateur peut s'il le souhaite coder sa propre stratégie et l'inclure dans l'application pour tester son comportement. Il doit pour cela créer un fichier `.java` héritant de certaines classes définies dans l'API (Application Programming Interface), compiler ce `.java` avec le `.jar` de jeu pour obtenir un `.class`. Il doit ensuite renseigner le chemin du fichier `.class` dans les réglages de l'application.

#### 4.2.6 IA dynamique

Au cours de la partie, les IA pourraient directement demander la parole au parrain pour réagir à une réponse donnée par un autre joueur ou pour donner une information. La demande de parole peut être représenté sous la forme d'un symbole d'exclamation "!" près de l'icône du joueur qui souhaite intervenir. Laissant la possibilité à l'utilisateur, en cliquant ou non sur le symbole "!", de lui donner la parole.

#### 4.2.7 Visualisation des connaissances

A tout instant de la partie, l'utilisateur peut accéder à la représentation des connaissances d'un des joueurs à ce stade du jeu.

Les connaissances seront représentées par une estimation des rôles qu'endosseront chaque joueur au yeux de l'un d'eux.

Ceci est représenté sous forme de tableau. L'interface laissera l'utilisateur passer d'un tableau à l'autre où chaque tableau représente les estimations des rôles des joueurs pour un joueur en particulier.

### 4.3 Éléments de validation du projet

Les éléments permettant de valider le projet sont d'une part, la livraison du logiciel opérationnel sous forme de fichier compilé exécutable, respectant les fonctions de service et de contrainte définies dans cette partie, d'autre part, la mise à disposition de l'ensemble de la documentation technique avec le cahier des charges, le rapport d'analyse et de conception et la manuel d'utilisation de l'application. La soutenance orale du projet sera accompagnée d'une démonstration de l'application.

## 5 Ressources

### 5.1 Ressources humaines

L'équipe de développement est composée de quatre étudiants. Elle est encadrée par deux professeurs.

- Encadrants :
  - Bénédicte Legastelois
  - Nicolas Maudet
- Etudiants :
  - Pierre-François Monville
  - Gilles Nouveau
  - Beatriz Rojas
  - Claire Salomé

### 5.2 Ressources techniques

Le langage de développement choisi pour réaliser le logiciel est Java version 1.8. L'interface graphique est développée en utilisant l'API JavaFx. Des bibliothèques compatibles peuvent aussi être éventuellement utilisées, pour faciliter par exemple la représentation des connaissances ou la logique des intelligences artificielles. L'organisation du code suit une architecture logicielle Modèle-Vue-Contrôleur. Le partage d'idées et de documents se fait avec Discord et Google Drive.

Le code source et tous les documents annexes sont disponibles sur le gestionnaire de version Github à l'adresse suivante :

<https://github.com/pfmonville/pandroide-mafia-de-cuba>

### 5.3 Planification du projet

Après avoir défini les différentes étapes du projet, l'équipe s'est reparti les tâches à effectuer tout en se fixant des contraintes de temps. Pour cette organisation, un diagramme de Gantt a été créé et régulièrement mis à jour en fonction de l'avancée du projet.



Created with Free Edition

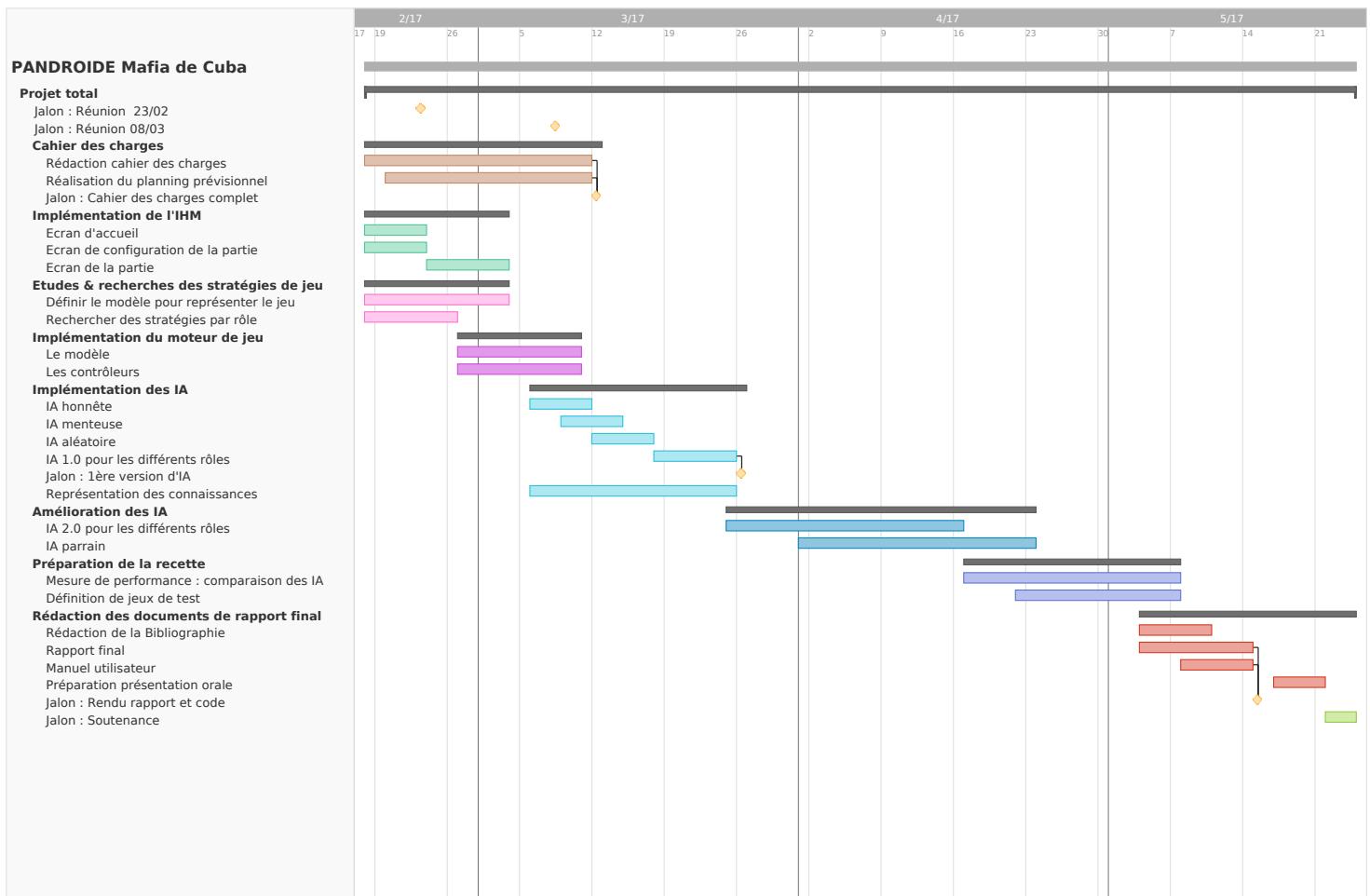


FIGURE 1 – Diagramme de Gantt

## Deuxième partie

# Analyse et conception

## 6 Logique de développement

Dès le début, l'équipe a eu à cœur de réaliser un projet évolutif, facile à reprendre et à comprendre. C'est pourquoi, tout le code est structuré selon le standard MVC (Model View Controller) afin d'organiser et de simplifier le développement.

Les modèles vont contenir ce qui se rapproche le plus d'entités réelles ou d'éléments consultables (un joueur, un rôle, un thème, une règle, une question, etc...); les vues vont s'occuper de gérer chaque élément de la GUI (Graphical User Interface); enfin les contrôleurs se chargent de récupérer les informations des modèles, de mettre à jour les vues, communiquent entre eux et représentent plus généralement le cœur de l'application (le contrôleur de jeu, le contrôleur d'un joueur, le contrôleur pour une stratégie, etc...). De plus, seule

une bibliothèque externe a été utilisée (*controlsfx*) afin d'afficher des popups dynamiques. Toute la partie graphique de l'application a été générée à partir de la bibliothèque interne de java, *javafx*.

## 6.1 Diagramme de classes

Voici les diagrammes de classes regroupant les trois packages principaux : Model, View, Controller

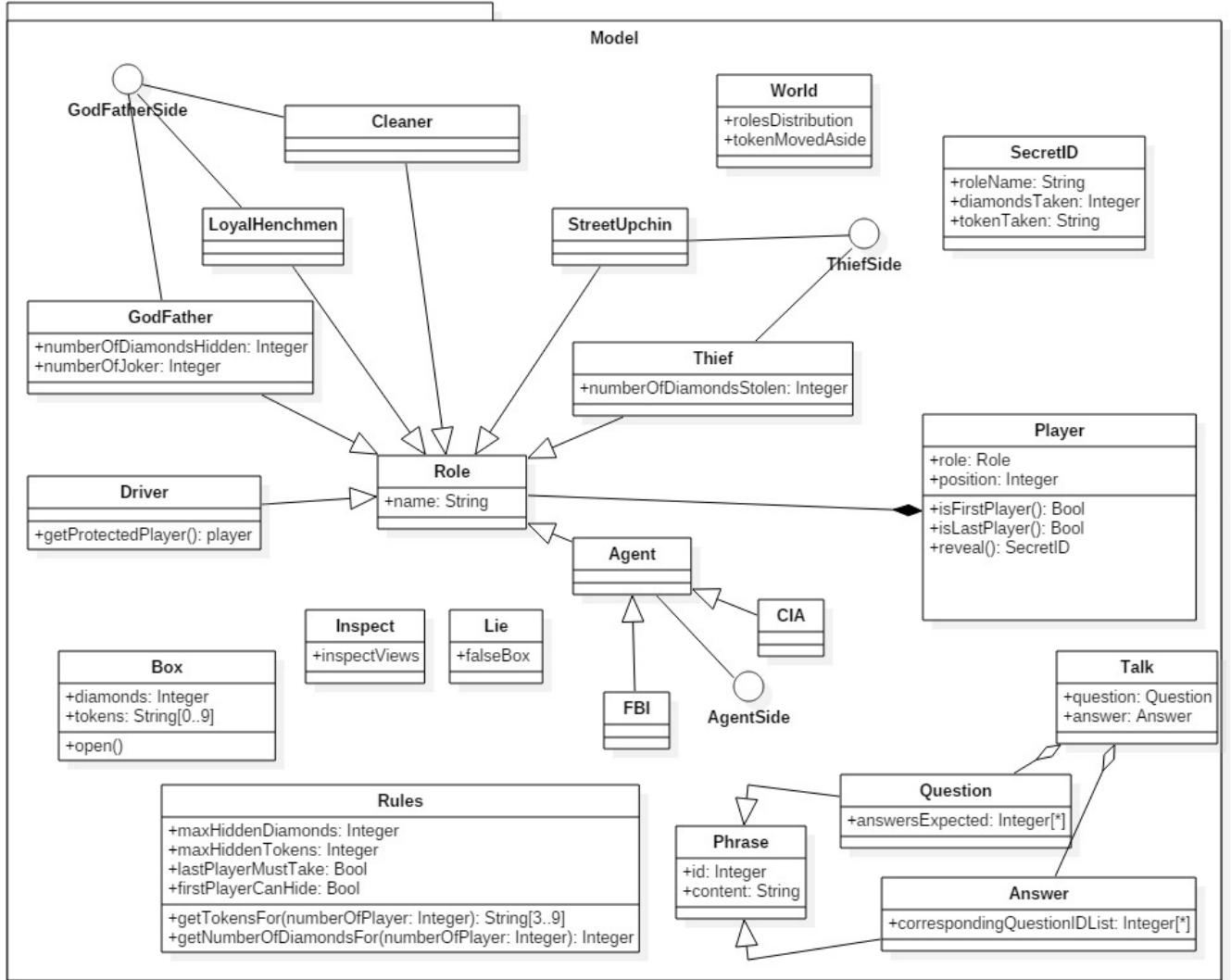


FIGURE 2 – Diagramme de classe des modèles

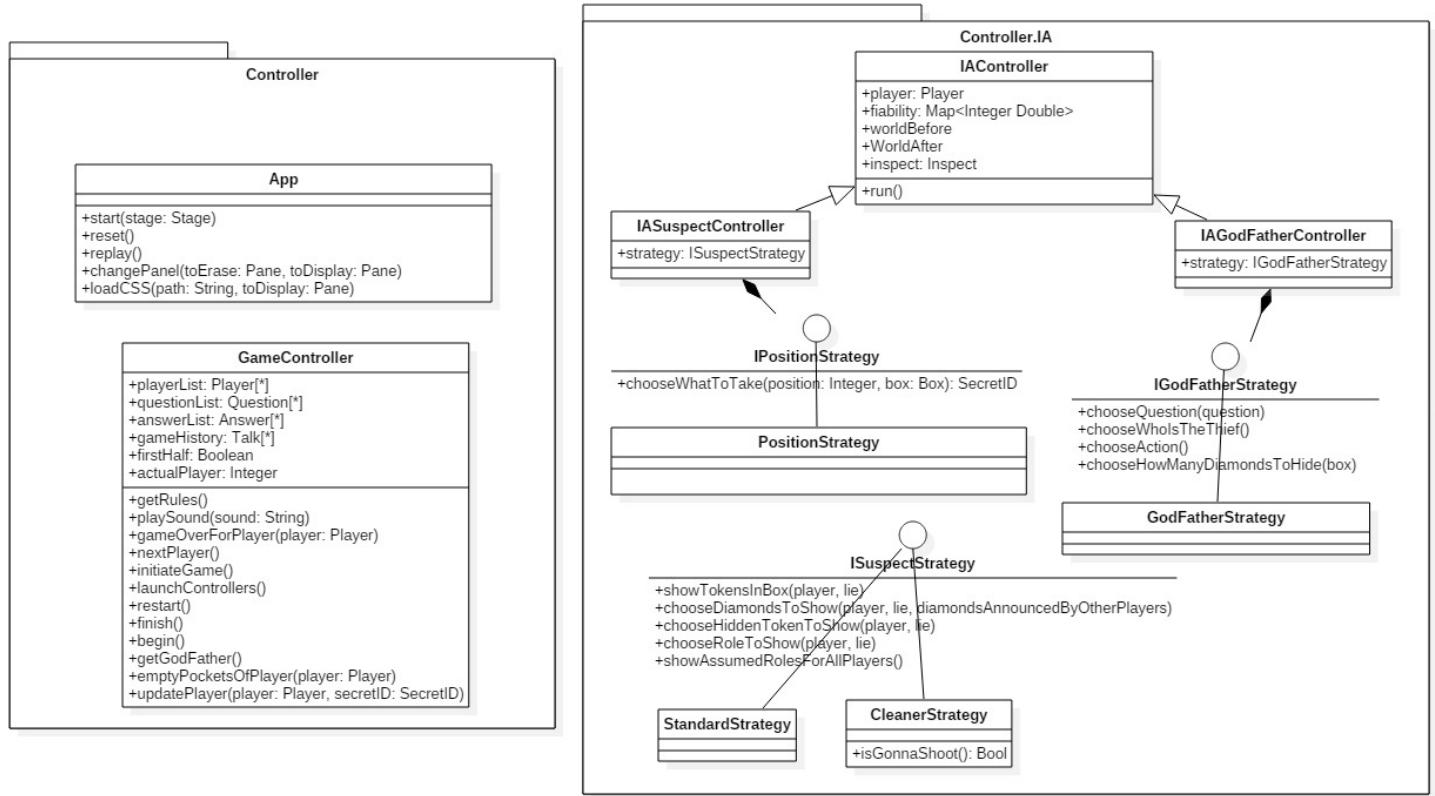


FIGURE 3 – Diagramme de classe des contrôleurs

## 6.2 Diagramme d'états

Voici le diagramme d'états décrivant le fonctionnement du cœur des contrôleurs. Tout le fonctionnement de l'application est résumé sur ce schéma :

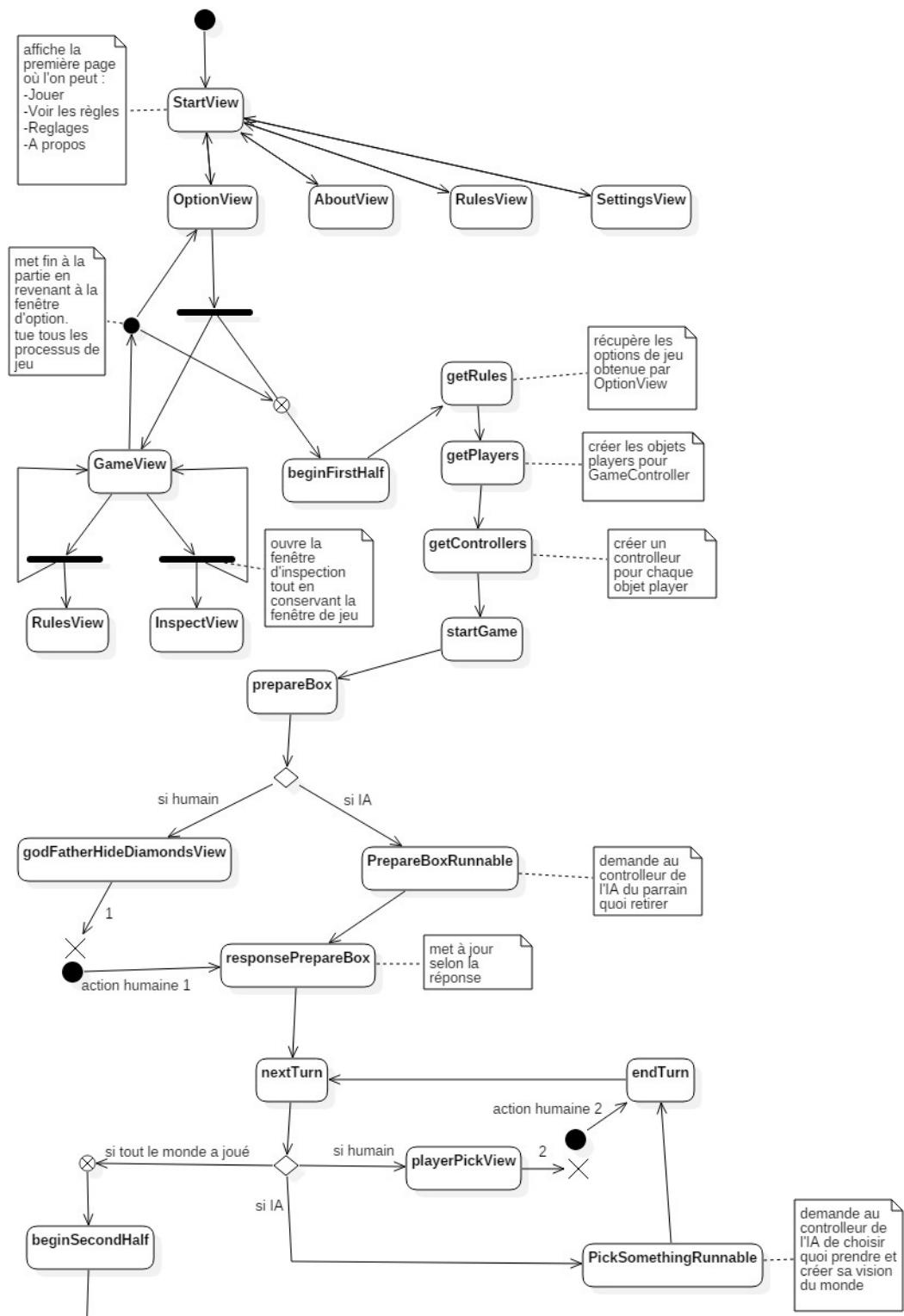


FIGURE 4 – Diagramme d'états partie 1

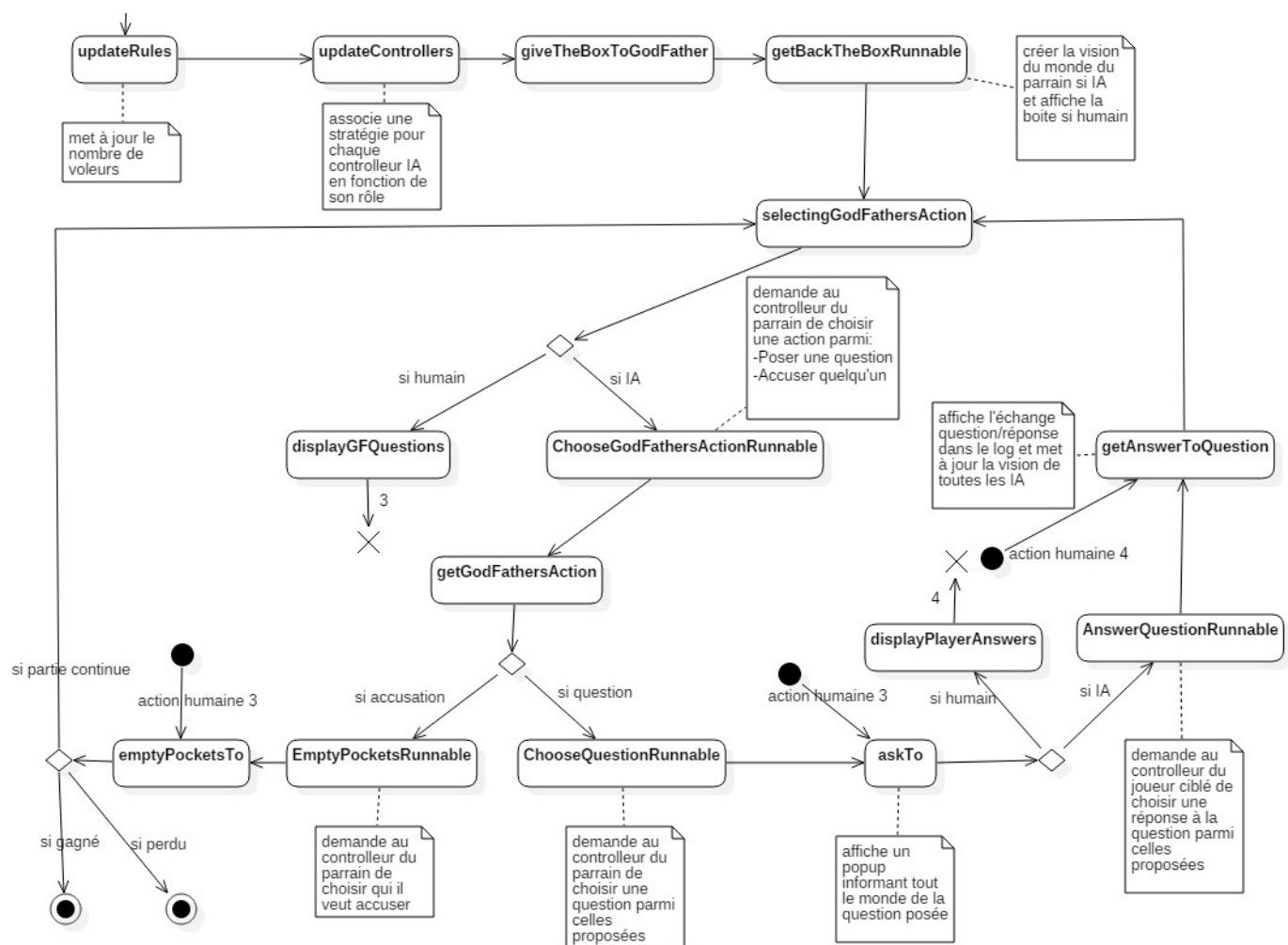


FIGURE 5 – Diagramme d'états partie 2

## 7 Fonctionnement des IA

### 7.1 Modélisation des connaissances des joueurs

Au début du projet, nous nous sommes interrogé sur la manière de modéliser, mettre à jour, exploiter et visualiser les connaissances d'une l'IA. Nous avons pensé à utiliser la logique modale pour représenter les certitudes et les croyances de l'IA vis à vis des annonces faites par les autres agents.

Nous avons également envisagé d'utiliser les modèles de Kripke pour la représentation et l'exploitation de ses connaissances. Cependant nous nous sommes rapidement demandé si cette modélisation était viable étant donné le nombre potentiellement élevé de mondes considérés et s'il était possible d'explorer et d'exploiter efficacement les mondes au fur et à mesure de la partie. Le manque de certitude sur la fiabilité des annonces des autres joueurs pouvait rendre l'exploitation des mondes assez complexe.

Notre but était que l'IA de chaque joueur soit capable d'identifier les différents rôles possibles des autres joueurs et d'écartier certaines possibilités au fur et à mesure que de nouvelles annonces sont faites. Ces informations pouvaient par la suite être utiles pour adapter les stratégies des joueurs au cours de la phase d'enquête. Ce sont également des connaissances importantes pour les Fidèles, qui ont pour but d'aider le Parrain à identifier les rôles de chaque joueur et plus particulièrement les voleurs.

Nous avons ainsi jugé utile de générer, lorsque l'IA reçoit la boîte, l'ensemble des configurations possibles concernant les joueurs la précédent. Elle doit ensuite effectuer le choix de son rôle puis générer les configurations concernant les joueurs suivants.

Nous avons choisi de séparer en deux les configurations possibles afin de réduire le nombre de mondes à générer ainsi qu'à parcourir après chaque annonce. Cette séparation est possible sans perte d'informations étant donné que les répartitions des rôles avant et après le joueur sont complètement indépendantes lorsque l'on connaît le contenu de la boîte.

Avant de réaliser la génération des mondes possibles, nous avons voulu estimer le nombre maximal de mondes générés pour les cas critiques dans une partie avec 12 joueurs, afin de déterminer si en pratique on pouvait calculer et exploiter ces mondes dans un temps raisonnable.

#### 7.1.1 Calcul des bornes sur le nombre de mondes avant et après

A chaque fois que la boîte est présentée à un joueur, celui-ci doit générer l'ensemble des mondes possibles en fonction de ce qu'il reste dans la boîte. Chaque joueur connaît l'état initial de la boîte et peut donc déterminer l'ensemble des diamants et des jetons à répartir parmi les joueurs et le parrain avant lui. Il décide de prendre quelque chose dans la boîte puis génère l'ensemble des mondes possibles pour les joueurs suivants.

Deux cas extrêmes se présentent ; celui du parrain qui reçoit la boîte à la fin et connaît donc l'ensemble des jetons et diamants à répartir parmi tous les joueurs, et le premier joueur qui doit générer l'ensemble des configurations possibles à répartir parmi tous les joueurs suivants. Nous allons nous placer dans ces deux cas afin de déterminer les bornes sur la quantité de mondes si l'on souhaite tous les générer.

Sachant que pour distinguer deux mondes, il faut qu'une personne n'ait pas le même rôle, la prise d'un ou plusieurs diamants pour un joueur ne génère pas deux mondes différents. En effet, on ne comptabilise que le rôle (fidèle, nettoyeur, agent, chauffeur, enfant des rues et voleur). Dans tous les exemples qui vont suivre, nous allons nous placer dans une partie à 12 joueurs.

Plaçons nous dans le rôle du parrain qui reçoit la boite. Deux cas extrêmes, soit la boite est vide (tous les jetons et diamants ont été répartis) soit la boite est pleine (seuls les diamants ont été pris). Ici plus la boite est vide, plus le nombre de monde augmente car il y a plus de rôles à combiner.

Dans un premier temps, on considère que le premier joueur ne cache pas de jetons, il y a donc 9 jetons à répartir parmi 11 joueurs (on ne compte pas le parrain).

Parmi ces 9 personnes, 2 d'entre elles seront des chauffeurs ; parmi les 7 qui restent, 2 d'entre elles seront des agents ; parmi les 5 qui restent, tous seront des fidèles.

On obtient ainsi :

$$\left( \binom{11}{9} \times \left[ \binom{9}{2} \binom{7}{2} \binom{5}{5} \right] \right) = 41580$$

Pour ceux qui restent, ils seront tous des voleurs excepté dans le cas où ils se trouvent après le dernier joueur à avoir reçu un jeton. Le premier des deux restant est nécessairement un voleur ; le dernier est donc soit un voleur soit un enfant des rues. Ce cas n'est possible que si l'un d'eux est tout à la fin. soit un remaniement de la sorte.

$$\left( \binom{10}{9} \times 2 + \left( \binom{11}{9} - \binom{10}{9} \right) \right) \left[ \binom{9}{2} \binom{7}{2} \binom{5}{5} \right] = 49140$$

Il reste à examiner le cas où le premier joueur cache un jeton.

Cette fois-ci il y a 8 jetons à répartir parmi 11 personnes. Il va falloir prendre en compte le rôle retiré par le premier joueur :

$$\binom{11}{8} \times \left[ \binom{8}{1} \binom{7}{2} \binom{5}{5} + \binom{8}{2} \binom{6}{1} \binom{5}{5} + \binom{8}{2} \binom{6}{2} \binom{4}{4} \right] = 124740$$

Pour ceux n'ayant pas de jetons, le premier est toujours un voleur et les deux restants se répartissent comme ceci : V,V ; V,E ; E,E s'ils sont tous les deux à la fin ou V,V ; V,E si seul l'un d'eux est à la fin. On obtient ce remaniement :

$$\left( \binom{11}{8} + \binom{10}{8} + \binom{9}{8} \right) \times \left[ \binom{8}{1} \binom{7}{2} \binom{5}{5} + \binom{8}{2} \binom{6}{1} \binom{5}{5} + \binom{8}{2} \binom{6}{2} \binom{4}{4} \right] = 165564$$

On obtient une borne maximale pour la générations des mondes du parrain qui est de 214704.

Regardons maintenant du point de vue du premier joueur, qui lui ne connaît pas la configuration utilisée à répartir parmi tous les joueurs. Dans le cas où le premier joueur décide de ne pas cacher de jetons ni d'en prendre. Sachant que a = nombre de jetons

agents,  $c$  = jetons chauffeur,  $f$  = jetons fidèle. On obtient :

$$\sum_{i=0}^9 \binom{10}{i} \left[ \binom{i}{a} \times \binom{i-a}{c} \times \binom{i-a-c}{f} \right]$$

Ce qui après simplification donne :

$$\sum_{i=0}^9 \binom{10}{i} \left[ \frac{i!}{a!c!f!} \right], a + c + f = i$$

Pour trouver une borne sup nous allons prendre  $a!c!f! > \lfloor i/3 \rfloor!$ , d'où cette nouvelle expression

$$\sum_{i=0}^9 \binom{10}{i} \left[ \frac{i!}{\lfloor i/3 \rfloor!} \right] = 1926101$$

Il faut rajouter la répartition voleur enfant des rues

$$\sum_{i=0}^9 \left[ (11-i) * \binom{10}{i} \left[ \frac{i!}{\lfloor i/3 \rfloor!} \right] \right] = 5742201$$

il reste le cas où le premier joueur prend un jeton :

$$\sum_{i=0}^8 \left[ (11-i) * \binom{10}{i} \left[ \frac{i!}{\lfloor i/3 \rfloor!} \right] \right] = 4532601$$

Le cas où il cache un jeton et prend des diamants est le même. Enfin, le cas où il cache un jeton et prend un jeton.

$$\sum_{i=0}^7 \left[ (11-i) * \binom{10}{i} \left[ \frac{i!}{\lfloor i/3 \rfloor!} \right] \right] = 1811001$$

le pire cas étant le premier, on obtient 5742201 mondes possibles.

Bien que le premier joueur a beaucoup plus de mondes à générer que le parrain (dans le pire des cas) ce nombre reste acceptable. Notamment pour une configuration à moins de 12 joueurs.

L'équipe a donc décidé de générer l'ensemble des mondes.

En pratique, les temps de calcul pour la génération concordent avec la complexité théorique. Le premier joueur met plus de temps que les autres. A partir du deuxième joueur le temps est beaucoup plus faible et ne fait que diminuer en passant de joueur en joueur. Le temps pour générer l'ensemble des mondes pour le premier joueur pour une partie à 12 reste cependant trop long pour être jouable en pratique. Cependant jusqu'à 10 joueurs, la génération du premier joueur est quasi-instantanée et ne représente que 108321 mondes. Enfin pour 11 joueurs, on obtient 853939 mondes.

### 7.1.2 Génération des mondes avant et après

Nous avons finalement décidé de représenter l'ensemble des connaissances de chaque joueur de la façon suivante :

- **World** : un monde possible considéré par l'IA. Il est constitué de :

- **rolesDistribution** : une liste représentant une répartition possible des rôles pour un ensemble de joueurs donné.

- **tokenMovedAside** : le jeton éventuellement écarté par le premier joueur dans cette répartition.

- **truthValue** : un **HashMap** qui associe à chaque joueur une valeur de l'ensemble  $\{-1, 0, 1\}$ . Initialisée à 0, cette valeur vaut 1 si cette répartition est possible d'après une annonce faite par le joueur concerné et vaut -1 sinon.

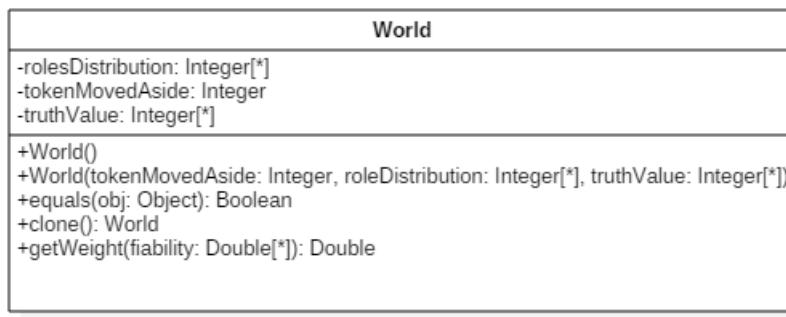


FIGURE 6 – Classe World

- **worldsBefore** et **worldsAfter** : l'ensemble des mondes possibles concernant les joueurs avant et après. Ce sont deux listes de **World**, initialisées lors de la phase de vol.

- **fiability** : un **HashMap** qui attribut à chaque joueur un degré de fiabilité compris entre 0 et 1. Cette valeur est initialisée à 0.5 et elle est mise à jour en fonction de la cohérence des réponses du joueur concerné.

- **notLoyalHenchman** : une liste contenant l'ensemble des joueurs dont on sait qu'ils ont menti au moins une fois

- **diamondsAnnouncedByOtherPlayers** : un **HashMap** qui associe à chaque joueur les quantités de diamants reçus et passés qu'il a annoncé au cours de la partie.

- **inspect** : un **HashMap** associant à chaque joueur une liste représentant la probabilité d'appartenance à chaque rôle.

Pour illustrer cela, prenons l'exemple du quatrième joueur d'une partie à 6 joueurs, où le premier joueur est le Parrain. Initialement la boîte contient un **Fidèle**, un **Chauffeur** et un **Agent**. Le quatrième joueur reçoit une boîte avec 8 diamants et un jeton **Fidèle**. Il prend le jeton **Fidèle** et passe donc une boîte contenant que des diamants. En début de partie, le deuxième joueur annonce qu'il est un **Chauffeur**. Les mondes générés, mis à jour après cette première annonce, sont les suivants :

**worldsBefore :**

tokenMovedAside	rolesDistribution		truthValue
	joueur 2	joueur 3	
Chauffeur	Agent	Voleur	{ 2 : -1, 3 : 0, 5 : 0, 6 : 0}
Chauffeur	Voleur	Agent	{ 2 : -1, 3 : 0, 5 : 0, 6 : 0}
Agent	Chauffeur	Voleur	{ 2 : 1, 3 : 0, 5 : 0, 6 : 0}
Agent	Voleur	Chauffeur	{ 2 : -1, 3 : 0, 5 : 0, 6 : 0}

**worldsAfter :**

rolesDistribution		truthValue
joueur 5	joueur 6	{ 2 : 0, 3 : 0, 5 : 0, 6 : 0}
Voleur	Voleur	{ 2 : 0, 3 : 0, 5 : 0, 6 : 0}
Voleur	Enfant des rues	{ 2 : 0, 3 : 0, 5 : 0, 6 : 0}

Pour calculer l'ensemble des configurations possibles à partir de l'état de la boîte nous avons séparé le problème en plusieurs cas de figures en fonction des règles du jeu.

Calcul de worldsBefore :

Lorsqu'un joueur reçoit la boîte, trois cas se présentent :

- Le nombre de jetons manquants est supérieur au nombre de joueurs avant lui. Nous pouvons donc déduire que le premier joueur a nécessairement écarté un jeton et que tous les joueurs avant le joueur courant ont pris un jeton personnage.

- Le nombre de jetons manquants est égal au nombre de joueurs avant lui. Nous avons alors deux possibilités :

- le premier joueur a écarté un jeton et une personne a volé des diamants.
- le premier joueur n'a pas retiré de jeton et aucun joueur a volé des diamants.

Nous remarquons que si la boîte contient 15 diamants, il ne peut pas avoir de voleur et nous pouvons traiter uniquement le deuxième cas. De même, si la boîte contient moins de 10 diamants, un joueur en a nécessairement volé et on traite uniquement le premier cas.

- Le nombre de jetons manquants est inférieur au nombre de joueurs avant lui. Nous pouvons déduire qu'au moins un joueur a volé. Nous devons tout de même traiter le cas où un jeton a été retiré ainsi que le cas où aucun jeton n'a été retiré.

Pour chacune de ces situations, nous avons utilisé la méthode **permutation** avec différents sous-ensembles de rôles en fonction du nombre de voleurs potentiels et du jeton qui a pu être écarté.

Calcul de worldsAfter

Pour générer les configurations possibles concernant les joueurs après le joueur courant, nous avons distingué quatre cas de figures :

- La boîte est vide. Les joueurs restants sont donc des enfants des rues.

- La boîte ne contient que des diamants. Les joueurs restants sont donc des voleurs et les deux derniers peuvent être éventuellement des enfants des rues.

- La boîte ne contient que des jetons. Les joueurs restants vont tous prendre un jeton, sauf éventuellement l'avant dernier joueur si tous les jetons ont été pris et le dernier joueur si également il ne reste plus de jetons ou s'il choisit de ne pas en prendre.

- Le boîte contient encore des jetons et des diamants. Pour ce cas, nous avons calculé le nombre maximal de voleurs possible, `upperBoundThieves`. Nous avons ensuite réalisé les différentes permutations avec des sous-ensembles de rôles contenant entre 0 et `upperBoundThieves` voleurs, en retirant du sous-ensemble le nombre de jetons respectif.

### 7.1.3 Mise à jour et poids des mondes

Certains des mondes peuvent être facilement écartés après quelques annonces. En effet, lorsque un autre joueur donne une information qui n'est possible dans aucun des mondes générés ou qui est incohérente par rapport à la boîte que l'IA a reçue et passée, on peut déduire que ce joueur ment, diminuer son degré de fiabilité et supprimer tous les mondes où il est Fidèle, s'ils existent.

Ainsi, si par exemple un joueur se situant avant l'IA annonce qu'il a passé moins de diamants de ce qu'elle a reçu après dans la boîte, la fiabilité du joueur interrogé est multiplié par un coefficient `contradictionCoeff` inférieur à 1 et le joueur est rajouté à la liste des non-fidèles. Cette détection d'incohérences et de mensonges est réalisé dans la méthode `checkLiar`.

Cependant, étant donné que la plupart du temps on ne pourra pas distinguer les vérités des mensonges, nous avons également attribué des poids aux mondes générés pour déterminer lesquels étaient plus probables.

L'IA de chaque joueur va donc calculer ces poids à partir de `fiability`, qui indique le degré de fiabilité des autres joueurs, et de l'attribut `truthValue` du monde concerné. Soit  $J$  l'ensemble des autres joueurs,  $f_i \in [0, 1]$  la fiabilité du joueur  $i$  et  $v_i \in \{-1, 0, 1\}$  la valeur de vérité de  $i$  pour ce monde. Le poids  $w$  est calculé comme suit :

$$w = \sum_{i \in J} f_i \cdot v_i$$

Étant donné qu'on diminue le taux de fiabilité d'un joueur pour chaque mensonge détecté, les poids des mondes où ce qu'il a annoncé est vrai vont également diminuer. En outre, les valeurs de vérité dans `truthValue` pour chacun des mondes sont mises à jour après chaque annonce lorsqu'aucun mensonge a été détecté.

Les poids des mondes sont par la suite utilisés dans l'objet `Inspect`, pour déterminer la probabilité qu'un joueur ait un rôle donné, pour chaque joueur et chaque rôle.

## 7.2 Stratégie du choix du rôle

La première phase du jeu est le vol des diamants par les différents joueurs. Chaque IA possède donc une stratégie qui lui permettra de se décider sur ce qu'elle va prendre.

Le contenu de la boîte évoluant, les stratégies mises en place dépendent de la position du joueur.

Nous avons alors créé quatre types de stratégie :

- *FirstPositionStrategy* : pour le premier joueur.
- *SecondPositionStrategy* : pour le second joueur.
- *MiddlePositionStrategy* : concernant les joueurs à partir du troisième jusqu'à l'avant dernier.
- *LastPositionStrategy* : la stratégie du dernier joueur.

Chaque joueur a à sa disposition tous les jetons personnage et les diamants que les autres auront laissés. Il peut donc décider aléatoirement d'être voleur ou d'être l'un des rôles disponibles, et de suivre la stratégie associée.

### 7.2.1 FirstPositionStrategy

Le premier joueur est le seul à savoir ce que le Parrain a écarté, et à pouvoir choisir parmi tous les rôles.

- S'il décide d'être voleur, il doit alors choisir combien de diamants prendre.

Dans le cas où il n'y a qu'un jeton *Fidèle* dans la partie, une de ses stratégies est de voler l'ensemble des diamants et de retirer le jeton *Fidèle* (ou *Nettoyeur*). Le Parrain n'aura donc aucun allié au cours du jeu, et en cas de défaite de ce dernier, le voleur sera forcément gagnant, étant le seul à avoir dérober des diamants.

Si le nombre de *Fidèle* n'excède pas 2, il peut alors prendre la moitié des diamants plus 1 (pour maximiser ses chances de victoire en cas de compétition avec un autre voleur) et écarter un jeton *Fidèle*.

Un autre cas de figure arrive quand le Parrain n'a pas écarté le maximum de diamants qu'il pouvait (ici 5) sur les 15 diamants initiaux. Le voleur peut alors dérober quelques diamants afin de laisser dans la boîte le nombre qu'il y aurait eu si le Parrain avait retiré tout ce qu'il pouvait (en l'occurrence 10). En écartant un personnage *Fidèle* ou *Agent*, il pourra ainsi se faire passer pour lui et éviter toute accusation.

En revanche, si le Parrain a enlevé le nombre maximum de diamants qui lui est autorisé, le risque de se faire attraper est un peu plus grand, donc le premier joueur préférera alors un autre rôle que celui de voleur.

- S'il décide de prendre le jeton *Chauffeur*, il sera alors dans le camp du Parrain, et envisagera alors d'éjecter un jeton *Agent*, pour faciliter la recherche des voleurs.

- S'il décide de jouer le rôle d'un *Fidèle* ou du *Nettoyeur*, là encore il pourra éjecter un *Agent*, ou aussi un *Chauffeur*, complice éventuel de ceux voulant faire perdre le Parrain.

- Enfin, si le joueur veut prendre le rôle de l'agent, il cherchera tout d'abord à éjecter le *Nettoyeur* de la partie s'il est présent, un *Fidèle* autrement. Les agents présents étant en compétition, une autre stratégie est aussi d'éjecter l'autre jeton *Agent*, afin d'avoir moins d'adversaires.

### 7.2.2 SecondPositionStrategy

Le second joueur est le plus à même de connaître le rôle du premier joueur, et potentiellement le nombre de diamants écartés par le Parrain. Grâce à ces connaissances, il en découvre différentes possibilités sur le rôle qu'il peut prendre.

Tout d'abord, si le joueur connaît l'identité du premier joueur :

- si le premier joueur est un voleur (il ne manque aucun jeton ou il y a moins de 10 diamants) :

Dans le cas où aucun jeton n'a été pris, le deuxième joueur n'a pas intérêt à voler à son tour. Il peut en revanche prendre le rôle de l'*Agent* et faire croire qu'il a volé, le vrai voleur allant normalement se déclarer innocent.

Une autre stratégie est de prendre un jeton *Fidèle*. Il pourra donc aider le Parrain à attraper au moins un des voleurs. Enfin, il peut choisir d'être le *Chauffeur* du premier joueur afin de l'aider à gagner.

Dans le deuxième cas où moins de 10 diamants sont présents dans la boîte et qu'un jeton a été écarté, le joueur peut aussi choisir son rôle en fonction du jeton retiré :

- le jeton manquant est un *Agent* : le Parrain ayant moins d'adversaires, le joueur peut alors choisir d'être de son côté et de prendre un *Fidèle*. Il peut aussi décider de prendre un *Chauffeur* ou un *Agent*, s'il en reste, car il sait déjà qu'il ne sera pas en compétition avec un autre. Une autre stratégie est de voler, les joueurs suivants pourront croire que c'est lui qui détient le jeton *Agent*.

- le jeton manquant est un *Chauffeur* : l'impact du *Chauffeur* n'étant pas très grand, le joueur peut réfléchir à son choix comme si le premier joueur n'avait rien écarté.

- le jeton manquant est un *Fidèle* : le Parrain ayant moins d'alliés au cours de la partie, les rôles de l'agent, chauffeur de voleur ou voleur sont privilégiés.

- si le premier joueur est un *Chauffeur* : le deuxième joueur sait alors que le premier se comportera comme un fidèle. Il peut alors suivre le mouvement en prenant un jeton *Fidèle* ou le deuxième jeton *Chauffeur* le cas échéant. Mais les rôles de l'agent et du voleur ne sont pas exclus de ses possibilités.

- si le premier joueur est un *Agent* : le deuxième joueur peut connaître cette information si les deux jetons *Agent* ont été écartés, ou si il y a le même nombre de diamants qu'à l'initialisation de la partie (ie 15). Dans ce cas, la stratégie qui consiste à prendre le deuxième agent n'est pas la meilleure, le joueur aura du mal à faire croire qu'il a volé. De même, être un voleur sera compliqué, l'agent risquant d'être accusé. Il préférera donc les rôles de *Fidèle*, et en particulier du *Nettoyeur*, ou du *Chauffeur*.

- si le premier joueur est un *Fidèle* : le premier joueur n'aura alors pas beaucoup d'informations au cours de la partie, il peut donc être intéressant pour le second joueur de voler des diamants ou de prendre le jeton *Agent*. Mais il peut aussi décider de devenir *Chauffeur* et donc de permettre au Parrain d'avoir un allié en plus des autres fidèles, ce qui est une stratégie préférable à celle de prendre un autre jeton *Fidèle*.

Mais le deuxième joueur peut aussi ignorer le rôle du premier, notamment s'il y a plus de 10 diamants et qu'il manque un jeton dans la boîte reçue. Il doit donc choisir ce qu'il prend en réfléchissant à ce qu'il se passerait si le premier est voleur ou le rôle pris :

- le premier joueur peut être voleur ou *Chauffeur* : le nombre de jetons *Fidèle* restants dans la boîte étant encore important, le second joueur prendra moins le risque de jouer le

rôle d'un agent ou d'un voleur.

- le premier joueur peut être voleur ou *Fidèle / Nettoyeur* : dans ce cas, le second joueur pourra tenter de prendre un jeton *Agent* ou de dérober des diamants, car le premier joueur ne sera pas d'une grande aide pour le Parrain. Mais il peut également prendre un jeton *Fidèle* au cas où le premier joueur est un voleur. Jouer le rôle du chauffeur est plus délicat, une mauvaise annonce mettant directement dans l'embarras son passager.

- le premier joueur peut être voleur ou *Agent* : la stratégie dominante est de prendre un *Fidèle*, pouvant avertir le Parrain. Le joueur peut choisir aussi de prendre un *Chauffeur* ou de voler, mais il prendra moins le risque de jouer le rôle d'un second agent.

### 7.2.3 MiddlePositionStrategy

Pour les joueurs ne se trouvant pas aux positions extrêmes, le contenu de la boîte peut être assez varié, ce qui peut impacter leur décision :

- la boîte reçue est vide : pas de choix possible pour le joueur qui sera *Enfant des rues*.
- la boîte ne contient que des diamants : le joueur étant nécessairement voleur, un meilleur choix pour gagner est de voler tous les diamants restants. Le joueur s'assure alors que les joueurs après lui seront *Enfants des rues* et de ce fait, qu'ils seront de son côté.
- la boîte contient des diamants et/ou des jetons : s'il reste des diamants et qu'il n'y a plus de jetons *Fidèle*, le joueur peut être tenté de prendre l'ensemble des diamants, étant donné qu'aucun des joueurs suivants ne sera du côté du Parrain. Sinon, il peut aussi voler la moitié des diamants plus un. Si un jeton *Agent* a été pris par un joueur précédent, ou qu'un joueur après lui sera obligé d'en prendre un (ie il y a plus de joueurs que de jetons restants) le joueur peut alors prendre le rôle du nettoyeur ou à défaut du fidèle. Mais il peut aussi tout à fait choisir le rôle du chauffeur ou de l'agent : son manque d'informations sur les mondes possibles ne doit pas l'empêcher de choisir ces jetons.

### 7.2.4 LastPositionStrategy

La stratégie du dernier joueur est la plus simple, étant donné que le joueur n'a réellement que deux possibilités : être *Enfant des rues* ou prendre ce qu'il reste dans la boîte. En effet, dans le cas où il recevrait une boîte vide, il n'aura pas d'autre choix de rôle. Pour tous les autres cas, sa connaissance des rôles des autres joueurs étant très faible il peut indifféremment prendre un jeton dans la boîte, des diamants restants ou rien. A noter que le dernier joueur préférera généralement être enfant des rues ou fidèle, ces rôles permettant de gagner grâce à d'autres joueurs. Les rôles où il se retrouverait seul sont plus durs à joueur dans son cas.

## 7.3 Stratégie du rôle public à montrer

Chaque joueur va devoir dire la vérité ou se faire passer pour quelqu'un d'autre. L'élaboration du choix du rôle à présenter aux autres c'est fait après l'analyse de parties réelles et par choix rationnel d'un agent selon deux degrés de complexité.

Le premier degré est la façon la plus direct d'obtenir son objectif; par exemple, un agent doit se faire accuser pour gagner et la façon la plus direct est de se faire passer pour un voleur.

Le deuxième degré de chaque joueur tient compte du graphe généré par toutes les stratégies au premier degré; par exemple un parrain tenant compte du premier degré va se méfier de ceux qui se font passer pour des voleurs, pensant qu'il s'agit d'agents, ainsi le voleur qui se fait passer pour un voleur simule donc l'agent qui se fait passer pour un voleur.

Il n'existe pas de troisième degré puisqu'on revient cela revient à jouer une stratégie au premier degré. L'ensemble de ces stratégies sont regroupées sur le schéma suivant :

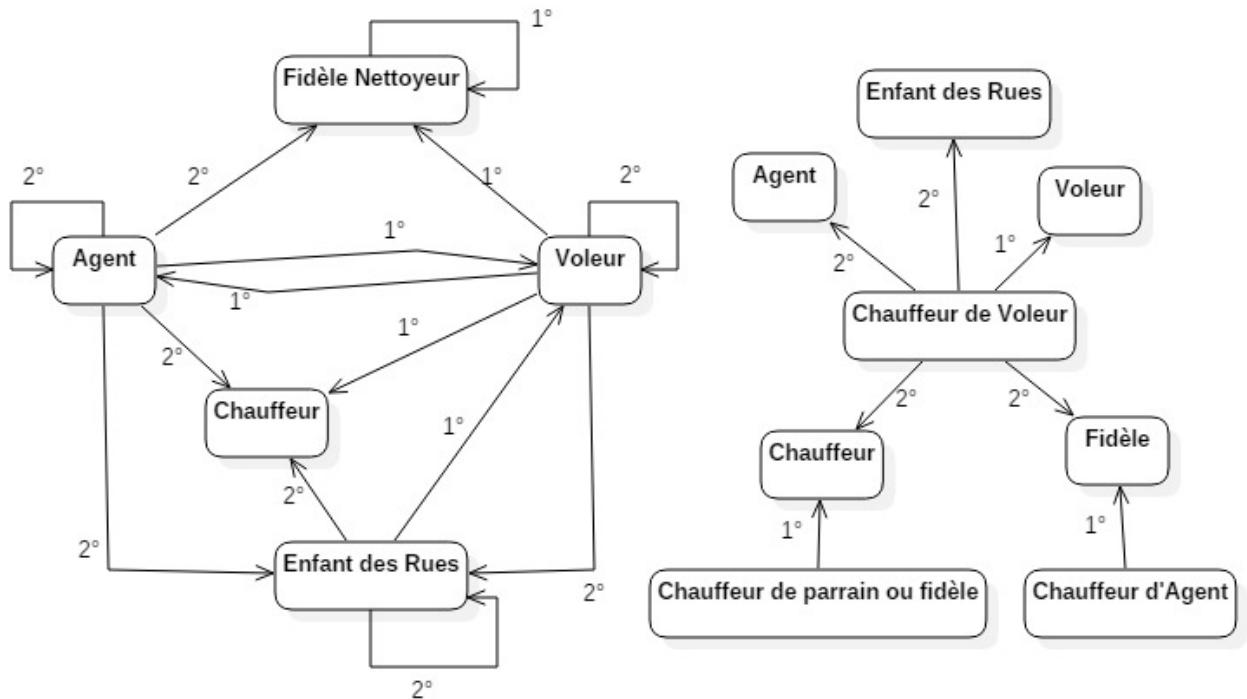


FIGURE 7 – Logique des stratégies au premier( $1^{\circ}$ ) et second( $2^{\circ}$ ) degré

## 7.4 Interactions entre l'utilisateur humain et les IA

Lors de nos tests en partie réelle avec le vrai jeu, nous avons observé que même si le Parrain mène globalement les échanges, les autres joueurs interviennent parfois quand ils le souhaitent pour faire une annonce globale ou alors que plusieurs groupes de discussion se forment pour débattre entre eux. Ce type d'interaction étant fortement complexe à implémenter, nous nous sommes fixé un cadre précis pour l'interaction entre le joueur humain Parrain et les IA.

Dans notre application, nous avons choisi de laisser le Parrain mener entièrement l'enquête. Il pose au fur et à mesure une question à un joueur précis et obtient pour chaque question posée une réponse du joueur ciblé. Cependant, dans un jeu basé sur la communication, les questions et réponses possibles peuvent être illimitées. Nous avons donc déterminé, à partir de nos retours d'expérience des parties réelles, une liste fixe de 17 questions, fermées et semi-ouvertes, dans laquelle le Parrain peut choisir pour interagir avec les IA. Ce sont les questions suivantes :

- Que contenait la boîte quand tu l'as reçue ?
- Que contenait la boîte quand tu l'as passée ?
- Combien de diamants contenait la boîte quand tu l'as reçue ?
- Combien de diamants contenait la boîte quand tu l'as passée ?
- Combien de jetons contenait la boîte quand tu l'as reçue ?
- Combien de jetons contenait la boîte quand tu l'as passée ?
- Quels rôles contenait la boîte quand tu l'as reçue ?
- Quels rôles contenait la boîte quand tu l'as passée ?
- Es-tu un... [rôle] ?
- Quel personnage es-tu ?
- Qui dois-je accuser selon toi ?
- Est-ce que tu sais quel est le rôle de cette personne ? [choisir la personne désignée]
- Selon toi quel est le rôle de cette personne ? [choisir la personne désignée]
- Est-ce que tu penses que cette personne a pris des diamants ? [choisir de la personne désignée]
- As-tu écarté un jeton ?
- Quel jeton as-tu écarté ?
- As-tu écarté le jeton ... [choisir le jeton]

Certaines de ces questions demandent à l'utilisateur de choisir une certaine valeur, comme le rôle, le type jeton ou la personne concernée par la question. Du fait du type de question, fermée et semi-ouverte, le format des réponses, bien définie, permet un traitement facile.

Les questions orbitent autour de 6 thèmes, à savoir : les diamants, les jetons, le contenu de la boîte en général (diamants et jetons), le rôle incarné par le joueur, l'avis de la personne sur un autre joueur et les questions pour le premier joueur concernant le potentiel jeton écarté. Une autre spécificité de certaines questions est la dichotomie entre la boîte reçue et passée.

## 7.5 Génération des réponses des IA

Pour répondre aux questions du parrain, les la plupart des joueurs ont besoin de se fabriquer un mensonge. Les fidèles ou le nettoyeur ont plutôt intérêt à donner des informations véridiques, cependant on parlera par la suite uniquement de mensonge ou d'informations fictives pour répondre aux questions. On sous entend alors que pour les joueurs du clan du parrain, le mensonge généré ou les informations fictives correspondent en fait à leur informations réelles.

Dans une première réflexion sur la manière dont l'IA peut créer son mensonge, nous avons pensé qu'elle peut le faire, après avoir reçu la boîte et une fois son rôle choisi. A partir du contenu réel de la boîte qu'elle avait reçue, elle se construit une fausse boîte, un faux rôle et un faux jeton écarté et si elle est première joueuse. Le mensonge est alors créé en une seule fois et il est ensuite utilisé tout au long pour répondre aux questions de façon honnête, comme le ferait un fidèle, mais en basant ses réponses sur le mensonge généré.

Cependant, nous avons remarqué que l'IA n'a besoin de créer son mensonge seulement au moment où elle est interrogée la première fois. Cela lui permet donc de bénéficier des réponses précédentes des autres joueurs, pour mettre à jour sa vision du jeu et ainsi améliorer le mensonge qu'elle va générer une fois interrogé.

De plus, nous avons remarqué que certaines questions concernent des domaines différents, comme par exemple les questions sur les diamants et les questions sur les rôles. Une réponse donnée dans l'un de ces domaines à peu d'impact sur l'autre, si l'on souhaite maintenir un mensonge cohérent. Notre idée est donc que l'IA débute avec un mensonge vide qu'elle fabriquera uniquement les informations fictives nécessaires pour répondre à la question qui lui est posé, afin de rester flexible sur les autres informations qu'on ne lui a pas encore demandé et de les générer en temps voulu avec potentiellement de nouvelles informations pouvant améliorer la cohérence de son mensonge.

La classe Java `Lie`, avec ses principaux attributs et méthodes, utilisée dans le code pour stocker les informations relatives au mensonge d'une IA est la suivante :

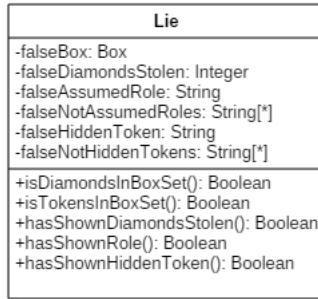


FIGURE 8 – Classe Lie

- `falseBox` : cet attribut est un objet de la classe `Box`. Elle représente le contenu de la boîte fictive qu'à reçu le joueur avec un `Integer` représentant le nombre de diamants et une liste de `String` qui indique les jetons cette boîte.

- `falseDiamondsStolen` : cet `Integer` stocke le nombre fictif de diamants volés.

- `falseAssumedRole` : `String` du rôle fictif que l'IA annonce aux autres joueurs.

- `falseNotAssumedRoles` : liste de `String` dans laquelle l'IA stocke les noms de rôle que l'IA dit ne pas être. Cette liste est mise à jour suite à la question "Es-tu un...?" avec un nom de rôle spécifié par le parrain. Si l'IA décide de répondre non à cette question, elle ajoute le nom du rôle en question dans cette liste.

Ainsi l'IA sait que dorénavant elle ne pourra plus annoncer être ce rôle pour assurer la

cohérence de son mensonge.

Suivant l'ordre des questions posées à l'IA, il est tout à fait possible d'avoir des élément dans cette liste sans que l'IA n'est encore définie son rôle fictif annoncé aux autre joueurs : **falseAssumedRole**.

- **falseHiddenToken** : **String** du rôle que le joueur annonce avoir écarté. Cet attribut n'est utilisé que par le premier joueur.

- **falseNotHiddenTokens** : liste de **String** des rôles que le premier joueur dit ne pas avoir écartés. Fonctionne sur le même principe que **falseNotAssumedRoles**. Cette liste est mise à jour après avoir répondu "Non" à la question fermée suivante : "As-tu écarté le jeton ... ?" avec un nom de jeton spécifique.

- **isDiamondsInBoxSet()** : retourne un **Boolean**, indiquant si le nombre de diamants dans la boîte fictive a déjà été initialisé.

- **isTokenInBoxSet()** : retourne un **Boolean**, indiquant si les jetons de la boîte fictive ont déjà été initialisés.

- **hasShownDiamondsStolen()** : retourne un **Boolean**, indiquant si l'IA a déjà annoncé le nombre de diamants qu'elle dit avoir volé.

- **hasShownRole()** : retourne un **Boolean**, indiquant si l'IA a déjà fait une annonce concernant son rôle et fixé l'attribut **falseAssumedRole**.

- **hasShownHiddenToken()** : retourne un **Boolean**, indiquant si l'attribut **falseHiddenToken** a été initialisé.

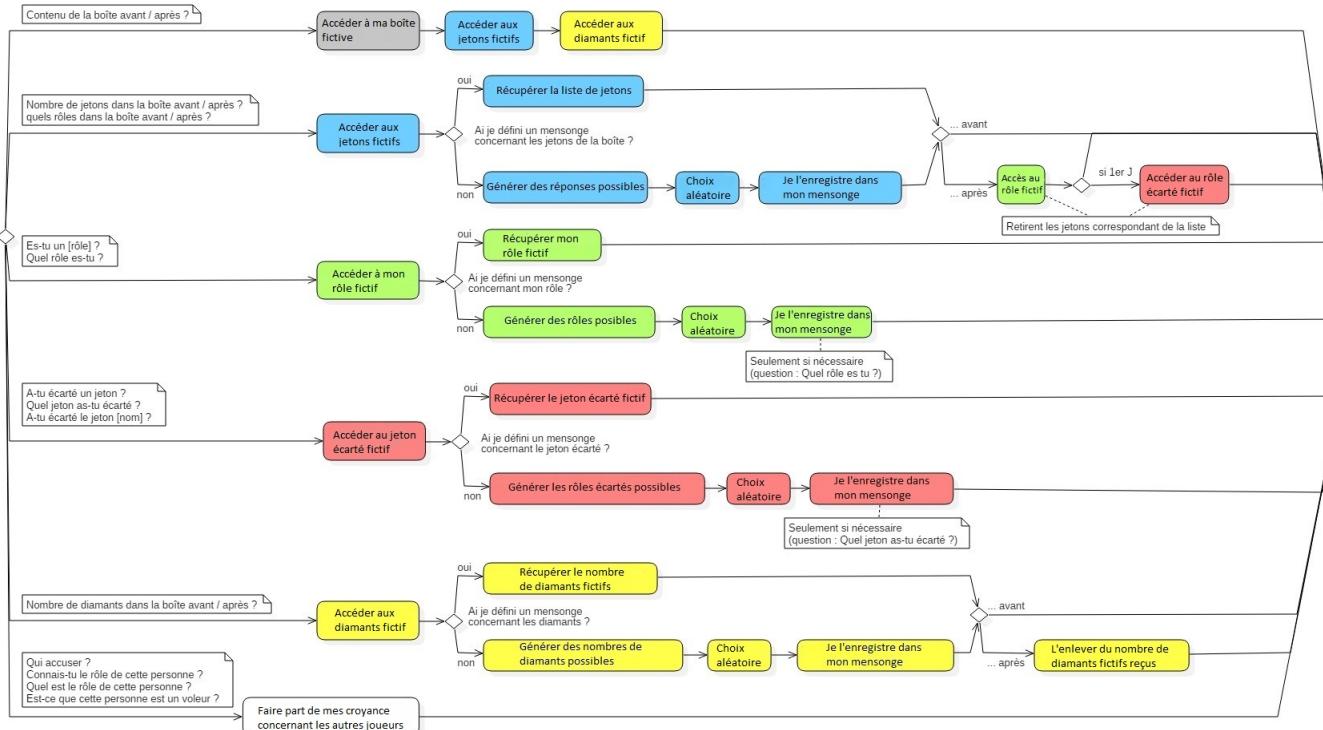


FIGURE 9 – Dérouler des différentes étapes pour la génération ou la mise à jour d'un mensonge en fonction de la question posée

## 7.6 Les Stratégies de rôle

Chaque IA va donc définir un mensonge qui sera la base de construction de ses réponses au Parrain. Le mensonge qu'elle devra suivre est le fruit d'une stratégie, qui varie en fonction du rôle de l'IA. Il y a en effet une pour chaque rôle.

### 7.6.1 Stratégie du voleur

Les objectifs du *Voleur* que nous avons identifiés sont les suivants : - Il ne veut pas faire porter les soupçons sur lui. Il va alors faire croire qu'il a un autre rôle, de préférence le *Fidèle* ou le *Nettoyeur*, pour avoir la confiance du *Parrain*, et que ce dernier désigne les autres joueurs à partir des informations qu'il donne. Il peut aussi se désigner comme étant *Chauffeur*, *Enfant des rues*, si sa position le lui permet, dans le principal but de ne pas se faire accuser. Sinon, il peut aussi jouer sur la peur du *Parrain* de désigner un *Agent* en se présentant comme tel.

- Il veut faire porter les soupçons sur les autres joueurs. Premièrement sur les autres *Voleurs*, surtout s'ils possèdent plus de diamants que lui, afin qu'il devienne le *Voleur* avec le plus de diamants en jeu. Ensuite, sur les *Fidèles*, *Nettoyeur* ou *Chauffeurs*, afin que le parrain perde ses jokers ou qu'il mette fin au jeu pour permettre au camp des *Voleurs* de

remporter la partie. Enfin, il doit faire en sorte que le *Parrain* ne désigne pas un *Agent* pour empêcher ce dernier de remporter seul la partie.

La stratégie implémentée dans le jeu s'inspire évidemment de points énoncés précédemment, de manière plus ou moins avancée.

- Pour les diamants présents dans la boîte : comme il veut se faire passer pour un joueur ayant pris un jeton, il doit annoncer le même nombres de diamants qu'il a reçu et qu'il a passé.

S'il est premier joueur, il n'a pas le choix d'annoncer le même nombre de jetons que celui donné par le *Parrain*. Par symétrie, s'il est le dernier joueur, il doit forcement dire que le nombre de diamants qu'il a reçu correspond au nombre de diamants qu'il a réellement donné au *Parrain*.

Étant donné qu'il a volé des diamants, il va devoir mentir soit sur le nombre de diamants reçus, soit sur le nombre de diamants passés, soit sur les deux valeurs en même temps tout en respectant l'égalité entre ces deux valeurs.

Le choix pour définir si le *Voleur* va mentir sur les diamants reçus ou passés va dépendre du nombre de jetons pris par les joueurs le précédent par rapport au nombre de jetons dans la boîte, qui seront éventuellement pris par les joueurs suivants. Si le nombre de jetons dans la boîte est inférieur au nombre de jetons pris, le *Voleur* va vouloir plus probablement faire porter les soupçons sur les joueurs le précédent et dire qu'il a reçu un nombre de diamants inférieur à ce qu'il a réellement reçu. De manière basique, la diminution du nombre de diamants reçus correspond au nombre de diamants qu'il a réellement volé.

L'idée est qu'en faisant porter les doutes sur les joueurs précédents, le *Parrain* aura plus de chance de désigner une personne avec un jeton et ainsi perdre la partie, faisant ainsi gagner les voleurs. Une amélioration possible et relativement simple mais qui n'a pas été implémentée aurait été de prendre en compte aussi le nombre potentiel d'agents avant et après le *Voleur*, pour orienter le choix afin de minimiser les risques de victoire des *Agents*.

Par symétrie, si le nombre de jetons dans la boîte est supérieur au nombre de jetons déjà pris, le *Voleur* va vouloir plus probablement faire porter les doutes sur les joueurs suivant en mentant sur le nombre de diamants passés qui correspond au nombre de diamants réellement reçus, comme s'il n'avait rien volé.

On rajoute un certaine probabilité de suivre le bluff concernant le nombre de diamants d'un des joueurs précédents. Le bluff est repéré si le nombre de diamants annoncé par les joueurs précédent ne correspond pas au nombre de diamants réellement reçu. La probabilité de suivre le bluff est d'autant plus faible que le joueur dont on a repéré le bluff est loin du *Voleur*, car cela augmente le risque que le mensonge soit remarqué par les joueurs intermédiaires. L'intérêt de cette stratégie est de former un bloc de joueurs donnant les mêmes informations, renforçant ainsi le sentiment de confiance qu'on peut leur accorder à ce groupe plutôt qu'à un joueur seul.

- pour son faux rôle : comme énoncé précédemment, le *Voleur* veut se faire plus probablement passer pour un *Fidèle*, ou un *Chauffeur* avec une probabilité moindre ou encore un *Agent* avec une probabilité encore plus faible. Les valeurs de probabilité écrites dans le code sont de 65% pour le *Fidèle*, 25% pour le *Chauffeur* et de 10% pour l'*Agent*.

Enfin dans le cas du dernier joueur, on ajoute la probabilité de 30% qu'il se fasse

passer pour un *Enfant des rues*. Dans ce cas on rééquilibre avec les autres probabilités précédemment définies pour sommer à 100%.

Pour savoir si le rôle qu'il dit avoir pris correspond à un jeton déjà pris avant ou alors qui était présent dans la boîte quand il l'a reçue, on utilise le même critère que précédemment sur la différence entre les nombre de jetons dans la boîte et le nombre de jetons pris au préalable.

- Pour les jetons présents dans la boîte : si le nombre de jetons dans la boîte est supérieur au nombre de jetons pris avant, il dit la vérité sur le nombre de jetons reçu, car son mensonge consiste à faire croire qu'il a pris l'un de ces jetons. Par contre si le nombre de jetons est inférieur à ceux pris avant et qu'il a déjà annoncé un rôle, il le rajoute à la fausse boîte qu'il a reçue. S'il n'a pas encore annoncé de rôle, il prend un des jetons pris précédemment qu'il rajoute à sa fausse boîte reçue. Par la suite, si on lui demande son rôle, il tirera au hasard un des jetons de cette fausse boîte créée.

- Pour le jeton écarté (s'il est premier joueur) : nous avons défini deux cas de figure plutôt basique. Si le *Voleur* a écarté un *Fidèle*, un *Nettoyeur*, un *Chauffeur* ou aucun jeton, il annonce avec une probabilité de 70% qu'il a écarté un *Agent* et avec une probabilité de 30% qu'il a écarté un *Chauffeur*. Ainsi il se fait passer pour un joueur souhaitant coopérer avec le *Parrain*.

L'autre cas de figure est la situation où il a écarté un *Agent*. Dans ce cas, il annonce n'avoir retiré aucun jeton pour se faire passer l'*Agent* qu'il a écarté et ainsi éviter de se faire accuser.

### 7.6.2 Stratégie du fidèle

Le fidèle doit aider le Parrain à débusquer tous les voleurs. De ce fait, le fidèle ne ment jamais, et sa stratégie consiste juste à dire la vérité sur le contenu de la boîte et ce qu'il pense des autres joueurs.

### 7.6.3 Stratégie du nettoyeur

Le nettoyeur se comporte exactement comme un fidèle, à ceci près qu'il a une stratégie supplémentaire, à savoir s'il doit tirer sur la personne accusée par le Parrain ou non.

### 7.6.4 Stratégie de l'agent

L'agent pour gagner doit se faire accuser par le Parrain. Sa toute première stratégie est donc de se faire passer pour un voleur. Il devra donc créer son mensonge en fonction de cette donnée :

- pour les jetons présents dans la boîte : l'agent dira la vérité sur ce qu'il a reçu, et dira qu'il a passé le même contenu au joueur suivant.

- pour son faux rôle : il annoncera derechef qu'il est voleur.

- pour les diamants présents dans la boîte : s'il a reçu moins de 10 diamants, il peut dire qu'il a reçu plus et donné le nombre qu'il a réellement passé ou annoncer qu'il en a donné moins, à condition qu'il ne soit pas le dernier joueur. En revanche s'il a reçu 10 diamants ou plus, l'agent préférera mentir sur ce qu'il a donné.

Par ailleurs, le joueur peut également décider de suivre le bluff de quelqu'un d'autre, en annonçant qu'il a bien reçu le nombre de diamants indiqué par un joueur précédent, mais toujours en décrémentant le nombre de diamants passés. La probabilité de suivre le bluff décroît avec la position du joueur précédent : plus le joueur est loin de l'agent, moins celui-ci va prendre le risque de le suivre.

- pour le jeton écarté (s'il est premier joueur) : si le joueur a effectivement écarté un jeton (autre qu'un agent) il peut dire la vérité sur ce jeton retiré. Si ce jeton était l'agent, il peut au choix dire la vérité ou affirmer qu'il a écarté un jeton *Fidèle*, ce qu'un voleur aurait tendance à faire. Si le joueur n'avait rien écarté, il peut encore une fois dire la vérité, ou faire croire qu'il a écarté l'agent, ce qui expliquerait sa disparition de la boîte, ou un autre jeton.

#### 7.6.5 Stratégie de l'enfant des rues

L'enfant des rues est un complice des voleurs. Il va de ce fait tout mettre en oeuvre pour que le Parrain n'accuse pas un voleur. Une stratégie première est de se faire accuser lui-même d'avoir volé, et donc de faire perdre le Parrain, ou au moins lui enlever un joker. Le comportement de l'enfant des rues est alors similaire à celui de l'agent :

- pour les jetons présents dans la boîte : le joueur dira la vérité sur ce qu'il a reçu, et dira qu'il a passé le même contenu au joueur suivant, ou au Parrain s'il est dernier.

- pour son faux rôle : il annoncera qu'il est voleur.

- pour les diamants présents dans la boîte : si l'enfant des rues est à la dernière position, il ne pourra mentir que sur le nombre de diamants reçus, ou éventuellement suivre le bluff d'un joueur précédent qui annonçait un nombre de diamants supérieur à 0. Par contre, s'il n'était pas le dernier joueur, cela signifie que la boîte qu'il a reçue était vide. Il peut alors librement mentir sur ce qu'il a reçu et/ou donné, ou encore une fois suivre la stratégie d'un joueur avant lui.

- pour le jeton écarté : cette question ne pourra pas lui être posée, car un enfant des rues ne peut pas être le premier joueur.

#### 7.6.6 Stratégie du chauffeur

Pour mener son rôle à bien, le chauffeur doit savoir ce que son passager a pris. Si le chauffeur est le premier joueur, il est alors considéré comme un fidèle et a donc la même stratégie définie précédemment. Pour toute autre position, le chauffeur aura un comportement plus naïf, en attendant de vraiment connaître le rôle du joueur qui le précède, qui consiste à dire la vérité, tout comme un fidèle l'aurait fait.

## 8 Améliorations

Parmi les fonctionnalités optionnelles, le niveau d'honnêteté est présent dans l'application, cependant les stratégies standards ne l'utilisent pas pour le moment. L'application permet d'ajouter ses propres stratégies pour tel ou tel rôle. Il suffit de suivre l'API en faisant hériter ou implémenter la bonne classe et de définir les fonctions nécessaires avant

de compiler le fichier avec le *.jar* de l'application. Il suffit de renseigner le fichier *.class* généré dans la fenêtre de réglage du jeu, l'application va détecter la stratégie externe et va charger celle-ci pour le rôle concerné plutôt que la stratégie standard.

La visualisation des connaissances est aussi implémentée, une fenêtre nous présente la vision d'un joueur en particulier quant aux rôles du reste des joueurs, on peut passer de la vision d'un joueur à un autre. La vision d'un joueur sur un autre est représentée par un pourcentage sur chaque rôle. Ce tableau est mis à jour en temps réel à chaque mise à jour de la vision des mondes des joueurs.

L'application actuelle permet à un utilisateur d'incarner le rôle du Parrain et de jouer contre des IA. Une évolution de l'application consisterait à implémenter une IA pour jouer le rôle du parrain. Cela permettrait de faire des parties avec uniquement des IA. L'un des intérêts serait notamment de configurer l'application pour effectuer un grand nombre de parties de façon autonome et d'ajouter sur les stratégies des IA une mesure de performance et des algorithmes d'apprentissage automatique (algorithme génétique, apprentissage par renforcement, réseaux de neurones). On pourrait alors observer quel type de comportement en fonction des rôles et de la position du joueur est le plus efficace et aussi améliorer la qualité des stratégies.

Avec l'ajout d'une IA pour jouer le rôle du Parrain, on pourrait permettre à un ou plusieurs utilisateurs d'incarner les autres joueurs autour de la table avec éventuellement d'autres IA aussi.

Une autre amélioration possible serait de permettre à une IA de demander au Parrain la prise de parole si elle estime avoir une information pertinente à diffuser aux autres joueurs. Cela pourrait permettre de rendre l'échange plus vivant et donc l'expérience de jeu plus agréable, mais aussi de permettre des stratégies plus avancées pour les IA avec cette nouvelle possibilité de jeu.

## Troisième partie

# Manuel d'utilisation

Au lancement du jeu, l'utilisateur arrive sur la page d'accueil où se trouve le menu principal :

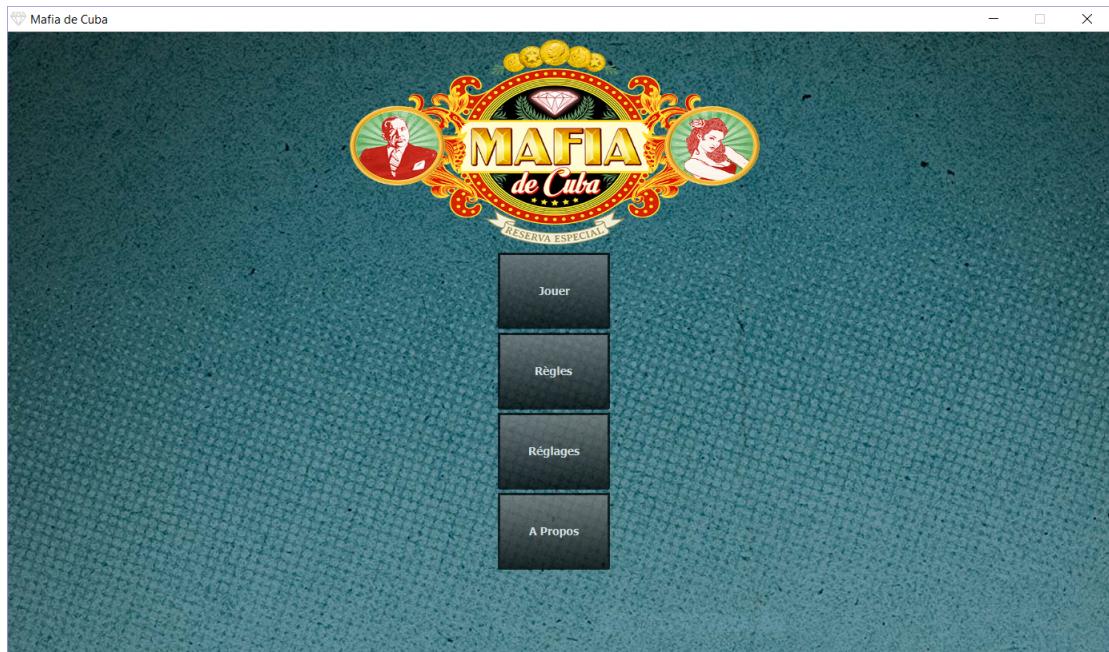


FIGURE 10 – Menu principal

Il peut alors lancer une partie (**Jouer**), accéder aux règles du jeu (**Règles**), paramétriser l'application, notamment avec l'ajout de ses propres stratégies (**Paramètres**) ou en savoir plus sur le développement du jeu (**A propos**).

En appuyant sur le bouton **Réglages**, l'utilisateur arrive sur une page d'options générales permettant de charger une stratégie personnalisée pour chaque rôle et de modifier le niveau d'honnêteté générale des IA.

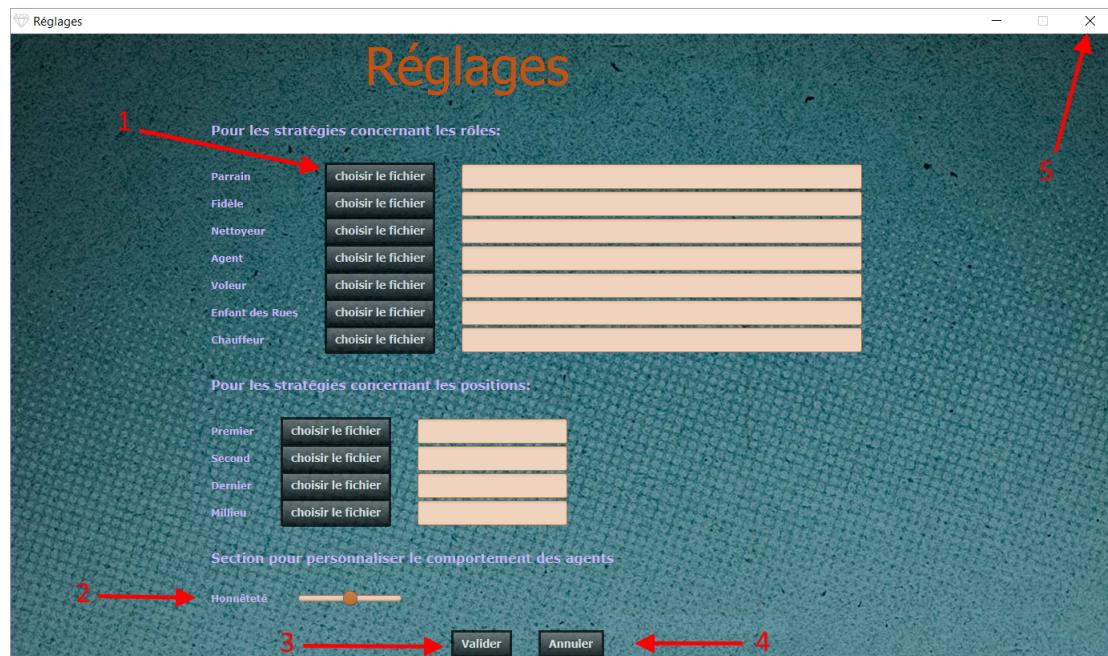


FIGURE 11 – Page d'options

Le joueur peut alors :

1 : ouvrir un explorateur de fichiers lui permettant de choisir le fichier .class correspondant à la stratégie qu'il souhaite charger pour ce rôle. Seuls les fichiers .class seront sélectionnables.

2 : changer le niveau général d'honnêteté des IA (que les stratégies peuvent prendre en compte)

3 : valider les changements actuels.

4 : revenir aux réglages par défaut.

5 : annuler tout changement opéré et revenir à la configuration précédente. (conserve toute stratégie déjà validée précédemment).

En appuyant sur le bouton **Jouer**, l'utilisateur arrive sur la page des options pour personnaliser sa partie :

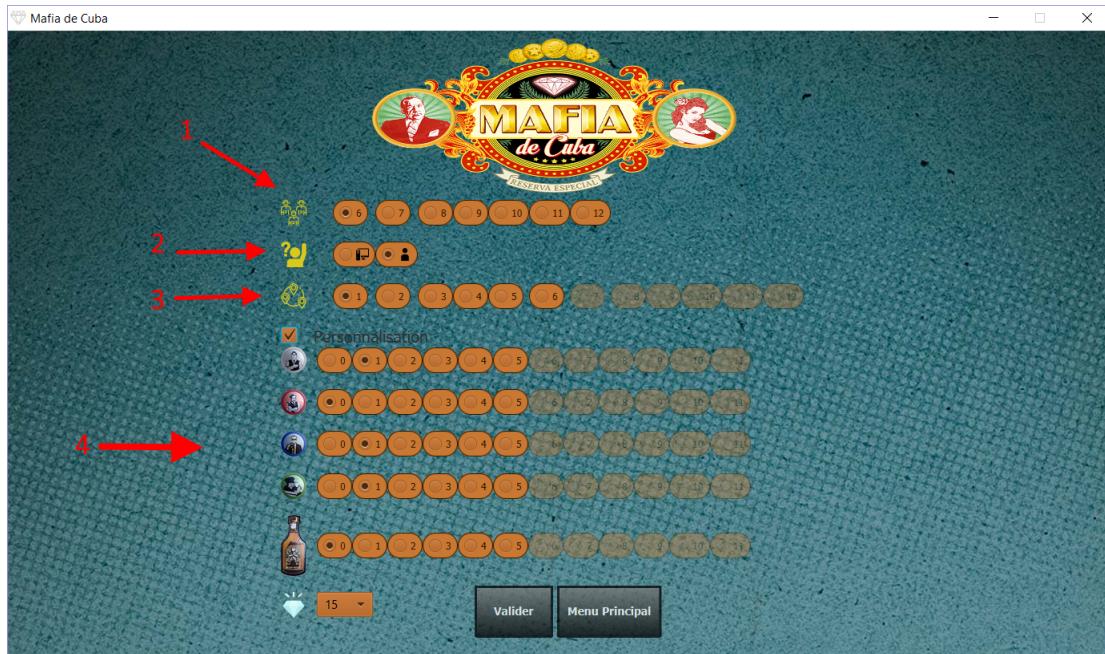


FIGURE 12 – Page d'options

Le joueur peut alors :

- 1 : sélectionner le nombre de joueurs
- 2 : choisir s'il veut participer à la partie ou regarder des IA jouer (nécessite la stratégie du parrain)
- 3 : décider de sa position dans le jeu. La position n°1 correspond à jouer le rôle du Parrain.
- 4 : personnaliser sa partie. Il peut changer les règles standards du jeu en modifiant le nombre de jetons de certains personnages, le nombre de jokers ou le nombre de diamants initialement présents.

Pour lancer sa partie l'utilisateur appuie sur le bouton **Valider**. Sinon il peut revenir au **Menu principal** en appuyant sur le bouton associé.

Une partie où l'utilisateur est le Parrain commence par l'affichage de cet écran :

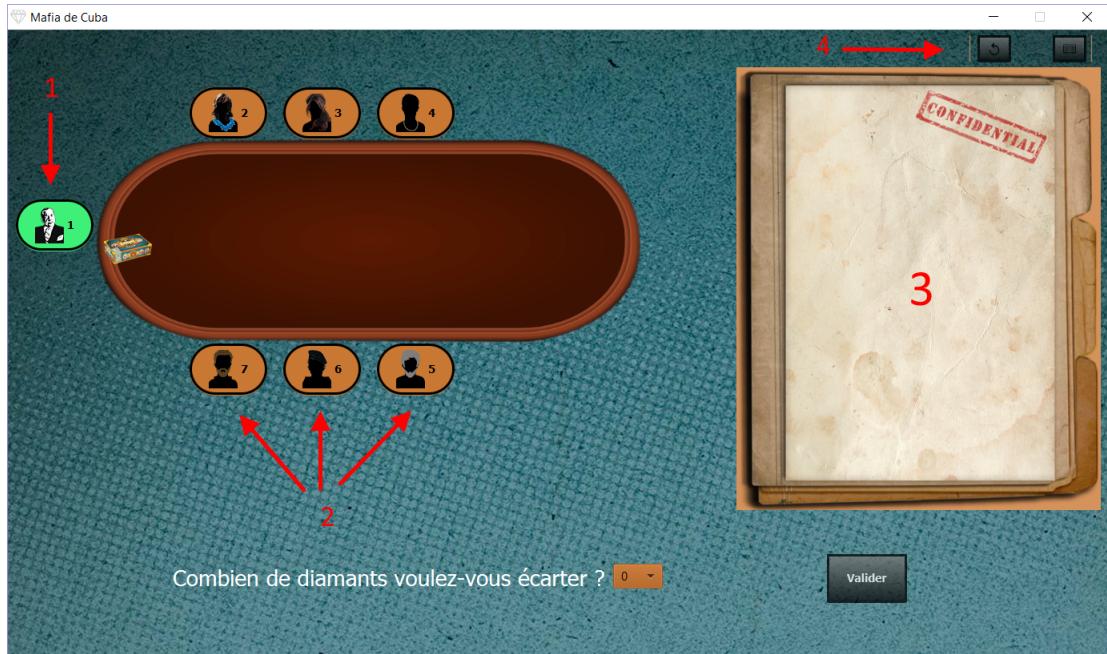


FIGURE 13 – Le Parrain écarte des diamants

Le joueur a alors l'occasion de découvrir l'interface du jeu :

1 : un bouton vert, indiquant où se situe l'utilisateur, avec un numéro de position associé. Ici, le joueur humain est le Parrain, sa position est la numéro 1.

2 : un ensemble d'autre boutons, représentant l'ensemble des joueurs de la partie, avec pour chacun un visage et une position, la plus grande position étant pour le dernier joueur.

3 : l'espace réservé à l'historique du jeu. Dès qu'une action se déroulera, elle sera automatiquement affichée dans cet espace, afin de permettre au joueur de se rappeler ce qu'il s'est passé tout au long de la partie.

4 : une barre d'outils lui permettant de revenir au menu des options défini précédemment ou de consulter les règles du jeu.

Durant cette première phase, le joueur doit sélectionner combien de diamants il veut retirer de la boîte, puis appuyer sur **Valider** pour commencer la phase d'enquête du jeu. Une nouvelle vue s'affiche alors :

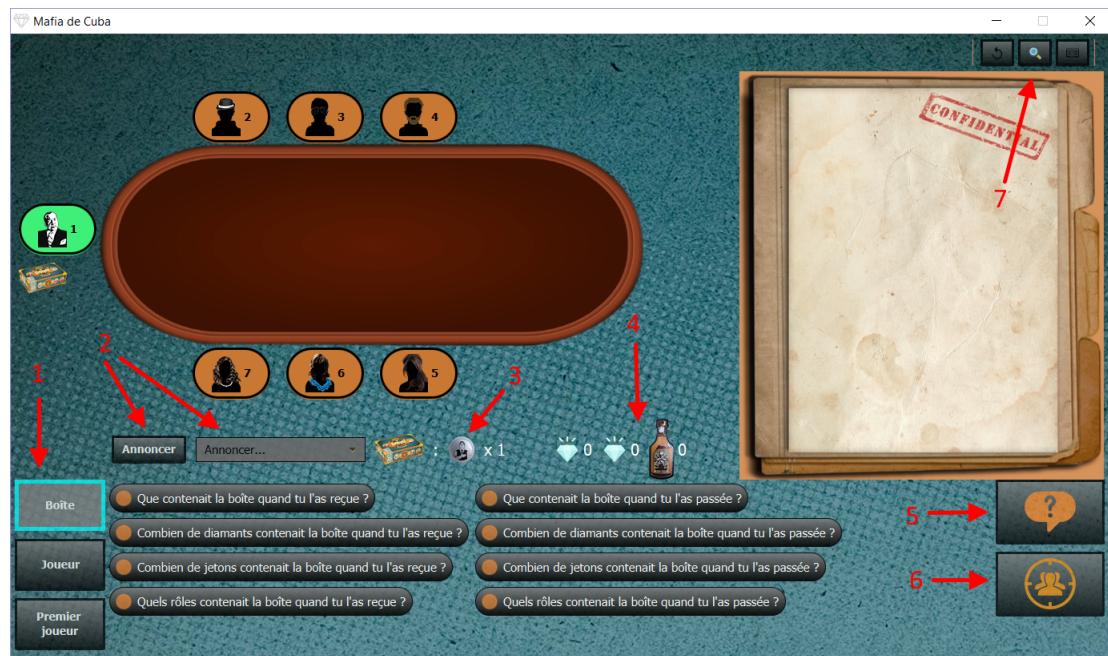


FIGURE 14 – Interface de la phase d'enquête

Sur cet écran se trouve plusieurs informations :

1 : des boutons de catégorie. En cliquant sur ces boutons, des questions différentes s'affichent, en fonction du thème souhaité : des questions sur le contenu de la boîte, sur les rôles des joueurs ou encore des questions spécifiques au premier joueur.

2 : un bouton permettant au Parrain de faire des annonces publiques sur ce qu'il a donné au premier joueur ou sur ce qu'il a reçu à la fin, en fonction du type d'annonce sélectionné .

3 : les informations sur ce que contenait la boîte quand elle est retournée au joueur.

4 : des rappels sur le nombre diamants que le joueur avait écarté lors de la première phase, ceux qu'il a récupérés, et sur le nombre de jokers restants.

5 : un bouton pour poser une question à un joueur autour de la table.

6 : un bouton pour accuser un des joueurs d'être un voleur.

7 : un nouveau bouton dans la barre d'outils, qui permet à l'utilisateur de suivre les mises à jour qu'effectuent les IAs sur leurs mondes possibles.

Pour interroger un joueur, l'utilisateur doit procéder dans cet ordre :

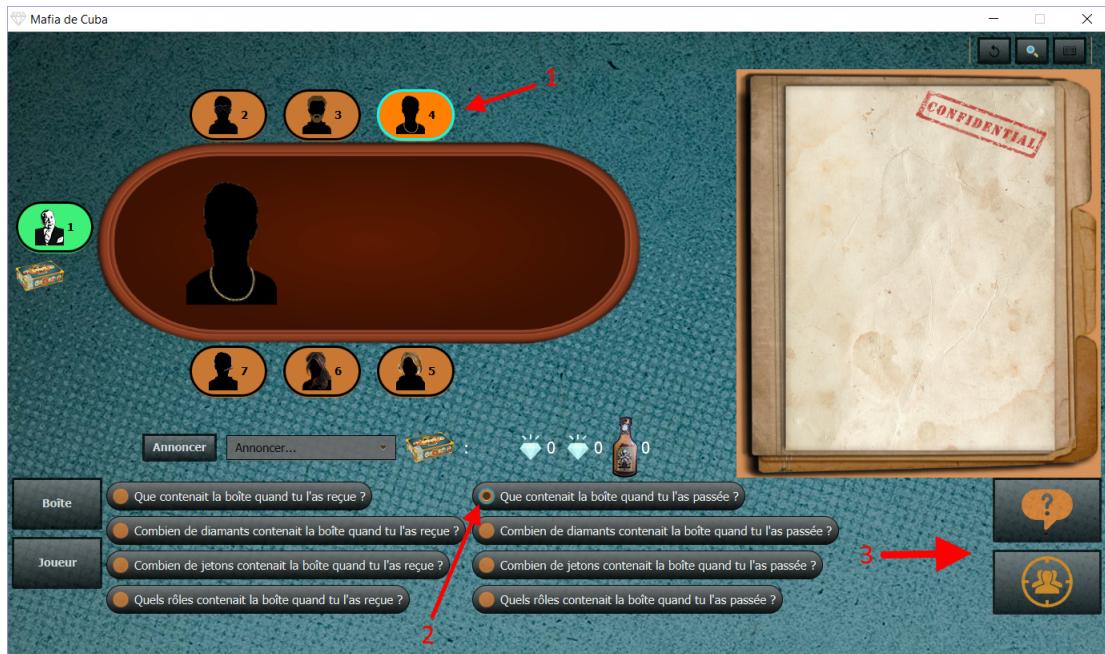


FIGURE 15 – Exécuter une action

- 1 : sélectionner le joueur voulu.
- 2 : cliquer sur la question souhaitée, si l'utilisateur veut interroger le joueur.
- 3 : choisir l'action entre poser la question sélectionnée, ou accuser le joueur ciblé.

Le jeu consiste alors à interroger et accuser l'ensemble des joueurs jusqu'à retrouver tous les voleurs. Lorsqu'un joueur est accusé par le Parrain, son rôle est révélé et remplace son image d'origine à son emplacement :

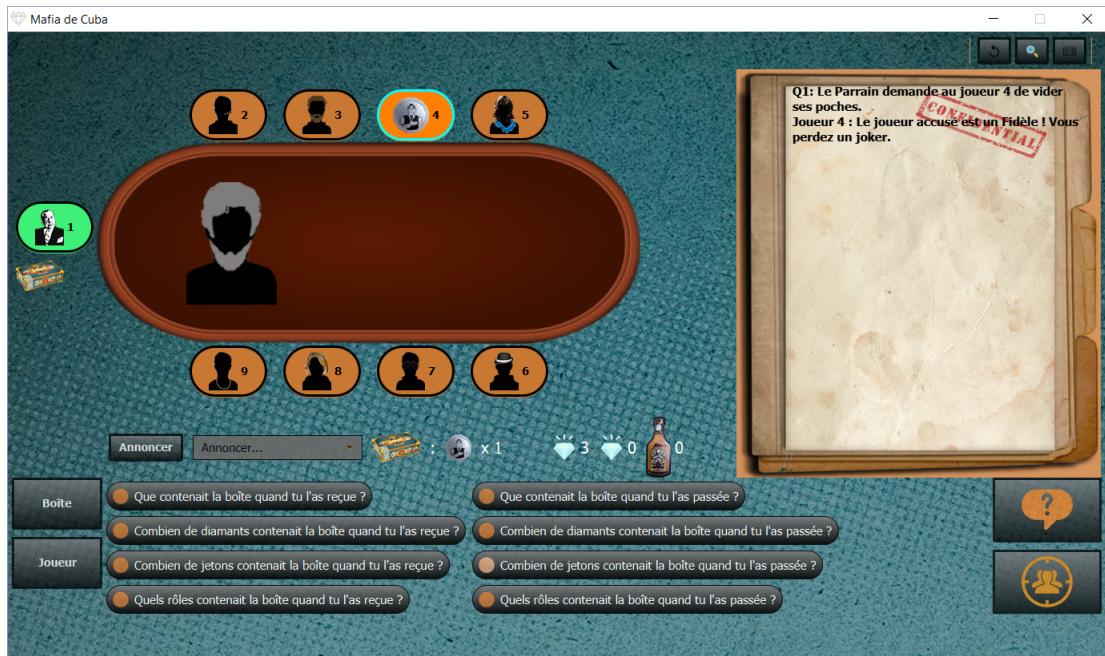


FIGURE 16 – Accusation d'un joueur

Ici, le joueur accusé était un fidèle camarade ! Son accusation entraîne la perte d'un joker, ce qui est rappelé dans l'historique du jeu. Si le joueur accusé n'est pas un voleur, le Parrain est libre de l'interroger à nouveau.

Une fois l'ensemble des voleurs retrouvé - ou des jokers épuisé - le jeu prend fin. L'utilisateur bascule donc sur l'écran de fin de partie suivant :



FIGURE 17 – Fin de partie - Le joueur a perdu

Tous les joueurs révèlent alors leur rôles, et s'ils sont voleurs, le nombre de diamants qu'ils avaient dérobés. L'historique du jeu affiche l'ensemble des joueurs ayant gagné, ainsi que des détails sur la répartition des rôles et le jeton écarté par le premier joueur. L'utilisateur peut ensuite retourner au **Menu principal** via le bouton dédié en bas à droite de l'écran.

Concernant l'API pour la création des stratégies. L'utilisateur, s'il souhaite implémenter une stratégie concernant un fidèle, un agent, un voleur, un enfant des rues, un chauffeur ou un nettoyeur, doit créer une classe qui implémente l'interface **ISuspectStrategy** ainsi que les 5 méthodes :

```
public HashMap<DiamondsCouple, Double> chooseDiamondsToShow(Player player, Lie lie, Map<Integer, DiamondsCouple> diamondsAnnouncedByOtherPlayers);
public HashMap<String, Double> chooseRoleToShow(Player player, Lie lie)
public HashMap<ArrayList<String>, Double> showTokensInBox(Player player, Lie lie)
public HashMap<String, Double> chooseHiddenTokenToShow (Player player, Lie lie)
private HashMap<String, Double> calculTokenResponseProbabilities(Player player, int lhNb, int dNb, int aNb, double lhProba, double dProba, double aProba, boolean isCleanerHere)
```

Chaque méthode renvoie un dictionnaire contenant comme clé une configuration envisageable et comme valeur la probabilité qu'il souhaite donner à son choix. On peut très bien n'envoyer qu'une configuration ou ne pas faire sommer les probabilités à 1 (une normalisation s'effectue par la suite par l'application).

Le constructeur de cette classe prend en paramètre un objet **Inspect** permettant de voir la vision de son joueur quant aux autres.

Enfin, si l'utilisateur souhaite coder une stratégie pour le parrain, il doit créer une classe qui hérite de **GodFatherBaseStrategy** et implémente l'interface **IGodFatherStrategy**.

Cette classe hérite de l'objet **Inspect** grâce à la classe mère, donc Il n'est pas nécessaire de faire un constructeur. Cependant l'objet **Inspect** n'est généré qu'après la première phase de jeu, le parrain ne dispose pas de cet objet lors de son choix sur le nombre de diamants à retirer. Donc une fonction de la classe mère est disponible pour tester si cet objet est disponible. **protected boolean isInspectSet()**

Dernièrement il y a 4 méthodes à implémenter :

```
public int chooseWhoIsTheThief()
public int chooseHowManyDiamondsToHide(Box box)
public int chooseAction() - 0 pour poser une question, 1 pour accuser
public Question chooseQuestion(ArrayList<Question> questions)
```

Il lui suffit de compiler son code avec la commande

```
javac -cp "chemin vers l'application en .jar" "le chemin vers son fichier java"
```

## 9 Bibliographie

Chiaki Sakama, Martin Caminada, Andreas Herzig. A formal account of dishonesty. Logic Journal of the IGPL, Oxford University Press (OUP), 2014, 23 (2), pp.259-294.

Benjamin Icard. Dynamique de la désinformation. Logique, pragmatique et théorie de l'esprit - Mémoire de master de philosophie. 2015.

Ronald Fagin, Joseph Y. Halpern, Yoram Moses and Moshe Vardi. Reasoning About Knowledge, The MIT Press. 1995.