Rapport - IA04

Le parlement version SMA

VILLAIN Benoit, KOVACIC Raphaël, LAVIOLETTE Etienne, GHITU Cristian

06/06/2016

Ce projet a pour but de nous faire implémenter un Système Multi-Agent utilisant une base de connaissances. Nous avons décidé de créer un jeu permettant d'évoluer au sein d'un parlement.

Sommaire

Introduction

Un Système Multi-Agents (SMA) comporte plusieurs agents qui interagissent entre eux dans un environnement commun. Ces agents sont des systèmes informatiques qui agissent de façon autonome et flexible pour faire ce pourquoi ils ont été programmés.

Dans notre cas, les différents agents ont pour but de faire fonctionner et d'animer un parlement. En effet, nous avons décidé de créer un SMA permettant de simuler le fonctionnement d'un parlement, de manière simplifiée. Ce qui est intéressant dans ce projet est que l'utilisateur n'est pas passif devant cette simulation, il participe activement aux différentes activités présentes au sein du parlement. Par exemple, l'utilisateur sera amené à voter des lois, en proposer, etc. Toutes ces actions auront un effet plus ou moins important.

Etant donné la forte importance de l'utilisateur dans cette simulation nous avons décidé de transformer cette "simulation" en un "jeu" où l'utilisateur devra devenir la personne la plus influente du parlement.

I - Conceptions

1. Conception du jeu

Le parlement et le monde extérieur.

Le parlement est l'environnement dans lequel les députés et le joueur vont interagir. Le monde extérieur lui est constitué (de manière simplifiée) du peuple et des entreprises. Ces deux entités vont jouer un rôle assez important durant ce jeu. En effet, ce sont eux qui vont juger si la "qualité de vie" et "l'économie" du pays se portent bien ou non (facteurs clés de la réussite du jeu).

Les députés

Tout d'abord, il faut savoir qu'un député est représenté par plusieurs caractéristiques dynamiques :

- Son appartenance politique.
- L'influence qu'il possède au sein du parlement par rapport aux autres membres.
- Sa crédibilité auprès du monde.
- Sa popularité auprès du peuple.
- Sa notoriété auprès des entreprises.

Toutes ces caractéristiques vont être amenées à évoluer suivant certaines règles préalablement définies. Pour être plus précis, celles-ci vont changer dès lors qu'une action importante sera faite durant le jeu. Par exemple, le fait de proposer ou de voter une loi changera augmentera ou diminuera certaines des caractéristiques des députés.

D'autres caractéristiques comme le charisme ou l'hésitation à voter pour ou contre une loi ne changeront jamais au cours d'une partie.

Un député ne pourra que : voter ou proposer une loi. Aucune autre action n'est possible.

L'utilisateur

L'utilisateur lui, possède exactement les mêmes caractéristiques qu'un député. La grosse différence avec les autres députés est qu'il peut faire plus d'actions. En effet, en plus de pouvoir voter et proposer une loi, l'utilisateur peut aussi :

- Faire un sondage lui permettant de savoir à quel niveau se situe la qualité de vie et l'économie du pays.

- Répandre une rumeur sur un des députés du parlement. Cette action est à double tranchant, car si celle-ci fonctionne, l'influence et la crédibilité du joueur augmentent fortement. Dans le cas contraire elles diminuent de manière significative.
- Parler aux députés pour avoir leur avis sur une loi donnée. (Pour ou Contre).
- Changer de parti.

Déroulement d'une partie

Il faut noter qu'un tour doit forcément débuter par la proposition d'une loi. La fin du tour elle se termine dès lors que le joueur humain vote pour une loi. Pour être plus précis, à tous les tours, une loi sera proposée soit par un député (2 tours sur 3) soit par le joueur (1 tour sur 3).

Si c'est au tour d'un député de proposer une loi alors, un député choisit au hasard propose une loi. Ensuite, tous les autres députés voteront pour ou contre cette loi automatiquement. Le joueur pourra alors décider, soit de faire une action parmi celles proposées, soit de voter pour la loi et donc mettre fin au tour de jeu.

Si c'est au tour du joueur de proposer une loi alors le joueur devra choisir parmi une liste de loi affichée à l'écran laquelle il veut faire passer. Ensuite, tous les autres députés voteront pour ou contre cette loi automatiquement ce qui mettra fin au tour de jeu.

Etant donné qu'à chaque tour une loi est votée, normalement, les différentes variables (des députés, du joueur et de l'environnement) devraient évoluer assez rapidement.

Victoire et Défaite

Pour gagner à ce jeu il faut que le joueur devienne la personne la plus importante du parlement. Par contre il faut que celui-ci fasse très attention à la qualité de vie et à l'économie de son pays car si celle-ci descendent en dessous d'un certain seuil, c'est la révolution assurée (défaite).

2. Conception SMA

L'agent Simulation

Cet agent a un rôle assez simple dans notre système. Il s'occupe juste de rythmer le jeu (la simulation). En d'autres termes, dès lors qu'il va recevoir un message de type REQUEST venant de l'utilisateur, il va faire démarrer la partie. A chaque fois qu'un tour se terminera (réception d'un message de type INFORM de l'agent médiateur), l'agent simulation débutera un nouveau tour en envoyant un message de type REQUEST avec la valeur du tour actuel à l'agent médiateur. Enfin, si celui-ci reçoit un message de type INFORM de l'environnement le prévenant qu'une de ses variables (qualité de vie ou économique) est descendue en dessous d'un certain seuil alors l'agent simulation arrête la partie et prévient l'utilisateur qu'il a perdu.

L'agent Médiateur

Contrairement au premier agent cet agent est un des plus compliqués. Il a pour but de s'occuper de toute la gestion du parlement dans son ensemble. C'est par lui que passe tous les messages envoyés par l'utilisateur et les différents agents. Nous allons maintenant voir les différentes tâches que celui-ci doit accomplir :

- Gérer le début d'un tour (réception d'un message REQUEST de l'agent Simulation) :
  > Envoyer un message de type PROPOSE à l'utilisateur lui demander de choisir une action à faire, de voter, etc.

- Gérer l'action que le joueur désire faire (réception d'un message ACCEPT_PROPOSAL)
  --> Renvoyer un message REQUEST si besoin de plus d'informations suivant l'action choisi.
  --> Le même schéma est utilisé si le joueur veut demander l'avis du parlement sur un loi précise.
  --> Si le joueur veut faire un sondage, le médiateur va tout simplement envoyer une REQUEST à l'agent Sondage qui va s'occuper du reste.
  --> Si le joueur veut changer de parti alors on envoie une REQUEST à l'AUtilisateur qui s'occupe du reste.

- Gérer les propositions et les votes de loi (réception d'un message ACCEPT_PROPOSAL)
  --> Si c'est au tour d'un député de proposer une loi, il va envoyer un message de type REQUEST à l'agent Loi pour qu'il s'occupe de gérer la proposition de loi par un député pris au hasard et le vote de cette loi (par les députés et/ou le joueur).
  --> Si c'est au joueur de proposer une loi, il va attendre de recevoir les informations de l'utilisateurs (QUERY_REF à l'AUtilisateur) avant de requêter KB pour récupérer la liste des lois que le joueur peut soumettre au vote (selon le parti du joueur). Il les transmet à l'AUtilisateur dans un message de type QUERY_IF. Il s'occupera ensuite de gérer la réponse (INFORM_IF)

venant du joueur accompagné de l'id de la loi à voter qu'il va « forwarder » à l'agent loi pour un vote.

- Gérer la fin d'un tour (réception d'un message REQUEST venant de l'agent Loi)
  --> Demande (REQUEST) à l'agent Simulation de passer au tour suivant.

L'agent Utilisateur

Cet agent représente le joueur. Il contient les caractéristiques de celui-ci et permet surtout de voir les différents messages qui lui sont envoyés. Cela permet notamment de vérifier le bon fonctionnement du jeu.

- Réception d'un message (PROPOSE de l'agent Médiateur) proposant de choisir une action parmi plusieurs actions.
- Réception d'un message (REQUEST de l'agent Médiateur) demandant des précisions sur l'action choisit.
- Réception d'un message (PROPOSE de l'ALoi) demandant de voter pour une loi.
- Réception d'un message (INFORM de l'ALoi) demandant de modifier ses caractéristiques.
- Réception d'un message (CONFIRM) proposant à l'utilisateur les partis éligibles au changement de parti.
- Réception d'un message (QUERY_REF de l'AMediateur) lui demandant de lui renvoyer ses informations personnelles (caractéristiques).
- Réception d'un message (QUERY_IF de l'AMediateur) contenant la liste des lois qu'il peut choisir.
- Contrôle en temps réel du niveau actuel du joueur afin de l'augmenter ou de le faire baisser si les conditions sont réunies.

L'agent Loi

L'agent loi s'occupe de gérer les propositions et les votes d'une loi.

- Réception d'un message (REQUEST de l'agent médiateur) demandant de faire proposer une loi (au vote ou au sondage) ou de faire voter une loi
  --> Si le message contient une loi proposée par l'utilisateur alors l'agent envoie un message (PROPOSE) à tous les députés pour qu'ils votent la loi ou donnent leur avis.

--> Si le message ne contient aucune loi, alors l'agent envoie un message (REQUEST) à un des députés pour qu'il propose une loi.

- Réception d'une proposition de loi d'un député (PROPOSE d'un agent député)
  --> Alors l'agent envoie un message (PROPOSE) à tous les députés pour qu'ils votent la loi (sauf celui ayant proposé la loi) ainsi qu'à l'utilisateur.

- Réception d'un vote favorable (proposition de loi ou demande d'avis) d'un député ou utilisateur. (ACCEPT_PROPOSAL)
  --> Mise à jour des votes.

- Réception d'un vote défavorable (proposition de loi ou demande d'avis) d'un député ou utilisateur. (REJECT_PROPOSAL)
  --> Mise à jour des votes.

- Lorsque tout le monde a voté
  --> Envoie de messages aux agents députés, à l'agent Utilisateur et à l'agent Environnement pour qu'ils modifient leurs caractéristiques suivant le résultat du vote. Envoi également d'un acquittement à l'agent KB pour qu'il retire la loi de la liste des lois possiblement proposables si celle-ci est effectivement votée et adoptée par le parlement.

- Lorsque tout le monde a donné son avis
  --> Envoie d'un message à l'agent médiateur pour lui signifier la fin de l'action.

Les agents Députés

Les agents députés peuvent seulement voter ou proposer une loi. Plus précisément, lors de :

- La réception d'un message demandant de proposer une loi (REQUEST de ALoi)
  --> Envoie un message (REQUEST à KB) pour récupérer une loi à proposer.
  --> Réceptionne la réponse de l'agent KB qui contient la loi à proposer.
  --> Répond avec un message (PROPOSE à l'ALoi) contenant la loi.

- La réception d'un message demandant de voter pour une loi (PROPOSE de Aloi avec conversation-id = « Proposition de loi »)
  --> Répond avec un message (ACCEPT ou REJECT PROPOSAL) suivant si le député vote pour ou contre la loi proposée

- La réception d'un message demandant de donner son avis pour une loi (PROPOSE de Aloi avec conversation-id = « Demande de sondage »)
  --> Répond avec un message (ACCEPT ou REJECT PROPOSAL) suivant si le député voterait pour ou contre la loi pour laquelle on a demandé son avis.

- La réception d'un message demandant de modifier ses caractéristiques (INFORM de ALoi)
  --> Modifie ses caractéristiques internes suivant le contenu du message reçu.

L'agent Sondage

L'agent Sondage, comme son nom l'indique s'occupe des demandes de sondage faites par l'utilisateur. Soyons plus précis :

- Réception d'un message (REQUEST de l'agent médiateur) demandant de faire un sondage sur l'environnement
  --> Envoie un message (REQUEST) à l'agent environnement pour connaître son état (variable qualité de vie et économie)

- Réception de la réponse (INFORM) contenant les variables de l'environnement.
  --> On informe l'utilisateur en affichant les variables à l'écran.

L'agent Rumeur

L'agent Rumeur est responsable de l'action de répandre des rumeurs au sujet des députés. Il fonctionne de la manière suivante :

- L'agent reçoit un message (REQUEST) de la part de l'agent Médiateur, ce qui déclenche l'action de répandre une rumeur ;

  --> Un comportement séquentiel est ajouté à l'agent Rumeur pour exécuter l'action, avant d'envoyer des messages (REQUEST) aux agents Députés et à l'agent Utilisateur pour connaître leurs variables d'influence, de popularité et de crédibilité. Ce comportement est constitué d'un comportement parallèle chargé de recevoir les caractéristiques des membres du parlement, d'un comportement atomique pour proposer une liste de députés à l'utilisateur et, enfin, un comportement générique chargé de traiter le choix de l'utilisateur ;

- L'agent envoie des messages (REQUEST) aux députés et à l'utilisateur pour connaître les trois caractéristiques mentionnées précédemment ;
- Les réponses (INFORM) sont récupérées et la liste des députés décrite par leur influence, leur popularité et leur crédibilité est transmise à l'utilisateur (PROPOSE) ;
  --> Les caractéristiques, de même que la liste, sont sérialisées en JSON avant d'être transmises.

- Le choix de l'utilisateur (ACCEPT_PROPOSAL) est récupéré, le message contenant l'indice du député dans la liste des députés connus par l'agent Rumeur. Ce-dernier calcule ainsi l'issue de l'action, en tenant compte des caractéristiques de l'utilisateur comparées à celles du député choisi. Suite au calcul, les valeurs de changement des caractéristiques seront transmises (INFORM) à l'agent Utilisateur et à l'agent Député choisi.

L'agent Environnement

L'agent environnement contient les caractéristiques du pays (Economie et qualité de vie). Cet agent permet de savoir comment se porte le pays et donc de savoir si le joueur est sur le point de perdre ou non.

- Réception d'un message (REQUEST de l'agent Loi) demandant de modifier les caractéristiques
  --> Modification des caractéristiques suivant le contenu du message.

- Réception d'un message (REQUEST de l'agent Sondage) pour connaître quelles sont les variables de l'environnement.
  --> Répond (INFORM) à l'agent Sondage avec un message contenant les variables demandées.

L'agent KB

L'agent KB est exclusivement en charge de la communication avec notre base de connaissances. Cet agent permet charger notre ABOX contenant nos diverses assertions. Il est aussi à sa charge la gestion du modèle JENA issu du chargement de nos assertions :

- Réception d'un message (REQUEST de l'agent Médiateur ou d'un agent Député) contenant une chaine de caractères correspondant au parti politique du demandeur. L'agent KB va interroger le modèle RDF JENA pour récupérer les lois éligibles à proposition pour le demandeur.
  Si la requête vient du Médiateur l'agent KB répond (INFORM) par une chaine JSON correspondant à 5 lois choisies au hasard et correspondants à son parti.
  Si la requête vient d'un député, KB sélectionne une loi au hasard et la renvoie (INFORM) dans une chaine JSON au demandeur.

- Réception d'un message (INFORM de l'agent Loi) contenant l'id de la loi votée à ce tour.
  KB s'occupe de mettre à jour le modèle JENA en passant la valeur de la propriété « is_voted » à TRUE dans le triplet dont le sujet est la loi porte l'id récupéré du message

La classe de l'agent KB est dotée de nombreuses méthodes que l'on pourrait qualifier de « GETTER » JENA. En effet, la manipulation du modèle RDF créé par JENA lorsqu'il charge les assertions de notre ABOX était difficile à manipuler. Nous avions besoin, tantôt de récupérer l'objet d'un triplet dans le sujet était fixé, tantôt de faire l'inverse. Dans un souci de découplage des fonctionnalités nous avons décidé de manipuler ce modèle RDF dans des fonctions à part.

3. Conception de la base de connaissance

La base de connaissances constitue l'un des piliers essentiels de notre réalisation. En effet, au-delà d'une exigence dans la consigne du projet c'est sur cette base que repose la dynamique du jeu. Dynamique qui est basée presque intégralement sur un mécanisme de vote de lois. C'est ainsi et tout naturellement que nous avons déterminé que notre base de connaissances serait une base de lois.

Une fois cette décision prise, il nous fallait déterminer les caractéristiques qui nous importaient dans une loi. Nous avons donc dressé la liste la liste des spécifications suivantes :

Loi = ID x Nom x Description x Effet sur le contexte économique x Effet sur la qualité de vie x Liste des partis politiques pouvant proposer cette loi.

Nous avons ensuite typé transformé chaque caractéristique en propriété dont nous avons typé l'objet :

```
law:id
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "An id for a law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "identification" ;
      rdfs:range rdfs:Literal .

law:name
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "A name for a law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "law name" ;
      rdfs:range rdfs:Literal .

law:desc
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "A description for a law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "law description" ;
      rdfs:range rdfs:Literal .

law:eco_effect
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "Give the economical impact of application of this law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "Economical impact" ;
      rdfs:range rdfs:Literal .

law:life_effect
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "Give the life impact of application of this law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "Life impact" ;
      rdfs:range rdfs:Literal .

law:politic_party
      a       rdf:Property , owl:ObjectProperty ;
      rdfs:comment "Political parties that can purpose the law" ;
      rdfs:domain parlement:loi ;
      rdfs:label "Political parties" ;
      rdfs:range rdfs:Literal .
```

Propriété d'une loi dans notre modèle

```
parlement:loi
      a       owl:Class , rdfs:Class ;
      rdfs:comment "A law." ;
      rdfs:label "Law" .
```

Représentation de la loi dans notre TBOX

En suivant notre TBOX, nous avons ensuite réalisé de nombreuses assertions dont voici un exemple :

```
@prefix parlement:    <http://xmlns.com/parlement/0.1/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix law: <http://xmlns.com/law/0.1/> .

parlement:loi1 rdfs:type law:Loi;
    law:id "1";
    law:name "L Comme Ri";
    law:desc "Pour faire baisser le chaud m'âge";
    law:eco_effect 10;
    law:life_effect -20;
    law:politic_party "Erudits", "Audacieux", "Altruistes";
    law:is_voted "false".

parlement:loi2 rdfs:type law:Loi;
    law:id "2";
    law:name "CP euhhhhhhh";
    law:desc "Pour que les jeunes balancent des pavés comme en 68";
    law:eco_effect 5;
    law:life_effect -10;
    law:politic_party "Altruistes", "Audacieux", "Sinceres";
    law:is_voted "false".

parlement:loi3 rdfs:type law:Loi;
    law:id "3";
    law:name "Pour promouvoir la merde audio MADE IN FRANCE";
    law:desc "De 8h à 20 h, 70% de la musique qui passent à la radio doit être de la musique française ";
    law:eco_effect 15;
    law:life_effect -15;
    law:politic_party "Fraternels", "Audacieux";
    law:is_voted "false".
```

Exemple de quelques assertions issues de notre base de lois

II - Réalisation

1. Interface graphique

Afin de rendre le déroulement du jeu plus plaisant, et d'aider l'utilisateur dans la manipulation des messages à envoyer, nous avons décidé de réaliser une interface graphique.

Cette interface a été réalisée à l'aide de JavaFX, choisi parce qu'il est le successeur de Swing qui a été abandonné et parce que depuis Java 8 en mars 2014, JavaFX est devenu l'outil de création d'interface graphique officiel du langage Java.

Nous avons décidé d'utiliser le pattern MVC. Pour ce faire nous fixons un package avec nos vues, des fichiers .fxml, un package avec nos contrôleurs liés aux fichiers .fxml, et notre modèle, des beans contenus dans un autre package, définissant les objets que l'on manipule. L'agent utilisateur est le seul à communiquer avec notre interface, il appelle des fonctions statiques contenues dans l'application principale de l'interface.

Le premier problème que nous avions à palier était le changement d'adresse IP que nous donnions aux agents secondaires pour se connecter au conteneur principal. Nous avons réalisé une fenêtre s'ouvrant au lancement de JADE et du conteneur principal. Cette fenêtre contient un champ texte, rempli par défaut par la valeur contenue dans le fichier propriétés de lancement des agents secondaires. On peut modifier cette valeur et appuyer sur le bouton pour écrire dans le fichier.

Ensuite, il nous fallait une interface pour voir les différents députés. Une fenêtre se lance au lancement de l'agent utilisateur, avec à gauche une liste déroulante avec nos députés, et à droite différents champs correspondant aux caractéristiques des députés. En bas de cette fenêtre, nous avons décidé de mettre une liste avec les différentes actions réalisées pour avoir un historique des actions réalisées.

Nous avons rajouté deux fonctions supplémentaires qui permettent au joueur de se prendre plus au jeu. La première est une visualisation des statistiques, le nombre de députés dans chaque parti .. La deuxième est une fenêtre avec différents achievements réalisables au cours du jeu.

On a ensuite, à chaque demande d'action de l'utilisateur, une pop-up qui s'affiche, avec les options possibles, et si précision de l'action nécessaire, une autre pop-up en plus s'affiche.

2. Tests et ajustements

Afin de créer un rendu qui soit exploitable lors de la soutenance de ce projet d'IA04 nous avons décidé de passer une partie importante de notre temps à modifier notre jeu dans sa globalité pour affiner le l'évolution des caractéristiques ainsi que de l'affichage.

En effet notre jeu se veut complexe au niveau de la gestion des caractéristiques à la fois du pays et de chaque député (utilisateur inclus). Afin d'éviter que notre jeu soit ou bien trop facile ou bien trop simple, nous avons décidé d'effectuer des ajustements de calculs lors d'une phase de tests finale.

Nous avons procédé en plusieurs étapes :

- Vérification de l'impact de chaque action (loi votée par l'utilisateur et acceptée par le parlement, loi refusée par l'utilisateur et acceptée par le parlement .. Ainsi que les nombreux cas de ce type).

- Étude de la longévité d'une partie standard.

- Évaluation de la pertinence des « niveaux » du joueur.

- Messages affichés sur l'interface graphique

- Messages sur la console pour témoigner de l'évolution et de la dynamique de notre jeu.

Ainsi nous avons décidé de borner les caractéristiques variables du joueur (humain) entre 40 et 60 à l'initialisation de la partie. Grâce à cela nous nous situons à la moyenne de la partie (20 au-dessus de la défaite et 20 en dessous de la victoire). La part d'aléatoire ajoutée par l'utilisation de la fonction aléatoire rend une partie plus ou moins dans une juste mesure.

Nous avons découpé les « réussites » du joueur en 4 niveaux :

- Moyenne des caractéristiques variables du joueur < 50 : niveau 0

- Moyenne des caractéristiques variables du joueur > 50 : niveau 1

- Moyenne des caractéristiques variables du joueur > 60 : niveau 2

- Moyenne des caractéristiques variables du joueur > 70 : niveau 3

- Victoire : moyenne des caractéristiques variables du joueur > 80

- Défait : moyenne des caractéristiques variables du joueur < 20

Nous avons finalement décidé de proposer un jeu qui soit adapté aux besoins de l'évaluation : beaucoup trop d'informations sont affichées dans la console et dans l'interface graphique. Notre jeu, pour l'évaluation est proposé en mode « triche » pour bien se rendre compte de l'évolution du joueur et de son environnement (ainsi que des messages échangées dans le SMA pur).

3. Outils et répartition du travail

Répartition et phases de travail :

La gestion d'une équipe de quatre développeurs n'étant pas chose aisée nous avons essayé, le plus possible, de nous répartir le travail également.

Nous avons découpé la réalisation de ce projet en trois phases distinctes.

Les trois semaines faisant suite à la constitution de notre groupe ont été dédiées à la phase de conception. Pendant celle-ci nous avons réalisé les tâches suivantes :

- Choix d'un sujet en accord avec notre chargé de TD
- Maquettes de conception successives afin de borner notre sujet selon nos envies et le champ des possibles
- Finalisation issue du consensus générale du jeu à créer

Cette étape fut la plus longue et la plus importante de notre projet. En effet nous avons, sciemment, pris le temps nécessaire à poser les bases d'un projet que nous souhaitions complet. Bien que la conduite d'un projet d'envergure comme celui qui nous était demandé n'est jamais linéaire, nous avons souhaité prendre le plus de décisions cruciales pendant cette phase de conception afin d'accord les quatre membres du groupe. Effectivement, chacun avait sa vision propre du jeu à réaliser et le partage des avis concernant un système multi-agent –pour lequel nous réalisions un premier projet important- n'étant pas toujours facile.

À la fin de cette phase de conception nous avions fixé plusieurs éléments :

- Les agents à utiliser
- Les messages échangés
- Les caractéristiques du jeu à implémenter (tour par tour, objectif du jeu, actions possibles de l'utilisateur…)
- Structure de l'ontologie
- Interface graphique à implémenter

Une fois la phase de conception actée et la solution à implémenter acceptée de tous nous sommes rentrés en phase de développement. Celle-ci s'est déroulée en plusieurs volés tout en se basant sur les éléments dégagés en phase de conception.

Le premier volet fut l'implantation de tous nos agents et les premiers messages échanges entre chacun d'entre eux. Cela constituait la brique de base de notre développement.

Le second volet fut le développement des actions complexes prenant nécessitant l'implémentation de la base de connaissances. Nous avons développement une TBOX représentant une Loi puis 50 assertions sur cette TBOX regroupées dans la ABOX. Pour lier cette base de connaissances aux éléments déjà développés nous avons codé l'agent KB et mis en place la gestion du modèle JENA dans l'application.

Une fois cette ontologie réalisée nous avons pu compléter notre structure de jeu en ajoutant les différentes actions la prenant en compte.

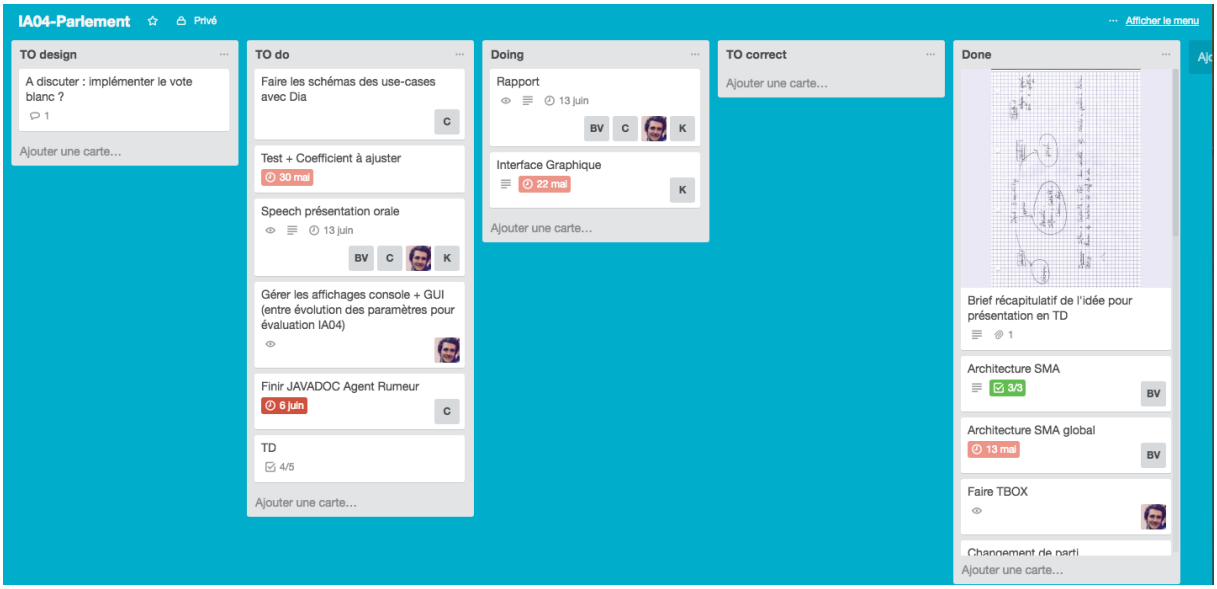
Dès lors que la base de connaissances était fonctionnelle, la suite du développement s'est faite selon la logique de déroulement du jeu.

Le troisième volet de développement fut celui de l'interface graphique. En parallèle du développement des actions de l'utilisateur, un membre du groupe s'est occupé de la réalisation de l'interface graphique de notre application avec toutes les difficultés que cela représentait. En effet, aucun des membres de notre groupe ne suivant en parallèle NF28, nous avons puisé dans nos connaissances personnelles pour la réalisation de cette partie du développement. Ce volet de développement fut l'occasion de nombreux échanges entre les concepteurs des différentes actions et le concepteur de la partie graphique.

La dernière phase de notre projet fut celle de finalisation : nous avons effectué le rassemblement de toutes les parties séparément développées (logique pure + partie graphique) ainsi qu'une longue phase de test. Nous avons pris conscience de la difficulté de créer un jeu « équilibré », qui ne tende ni trop rapidement vers la victoire, ni trop rapidement vers la défaite. Nous avons ajusté les coefficients représentants les parts d'aléatoire prises en compte lors des différents calculs dans notre code.

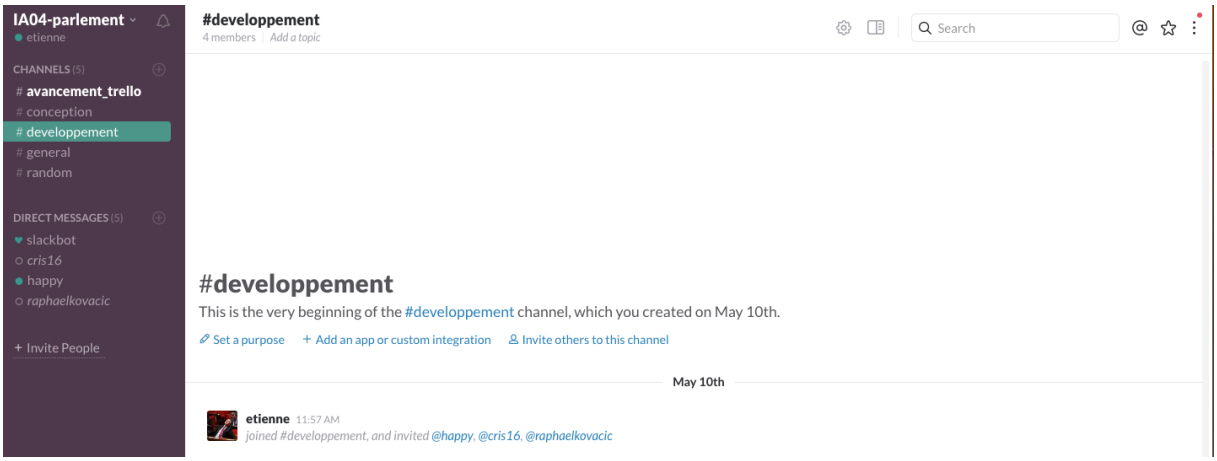
Outils :

Trello : Afin de discrétiser, d'atomiser et de se répartir les tâches durant les différentes phases de notre projet nous avons utilisé le puissant outil de gestion de projet Trello. Celui-ci se présente sous la forme d'une table représentant, en colonne, les différentes phases de développement du projet. Chaque tâche est représentée par une carte et est déplacée de colonne en colonne selon son état actuel. Trello permet d'affecter des membres à des tâches, de fixer des deadlines à chacune d'entre elles. Ce fut un précieux outil à la conduite de ce projet.



Outil de gestion de projet Trello

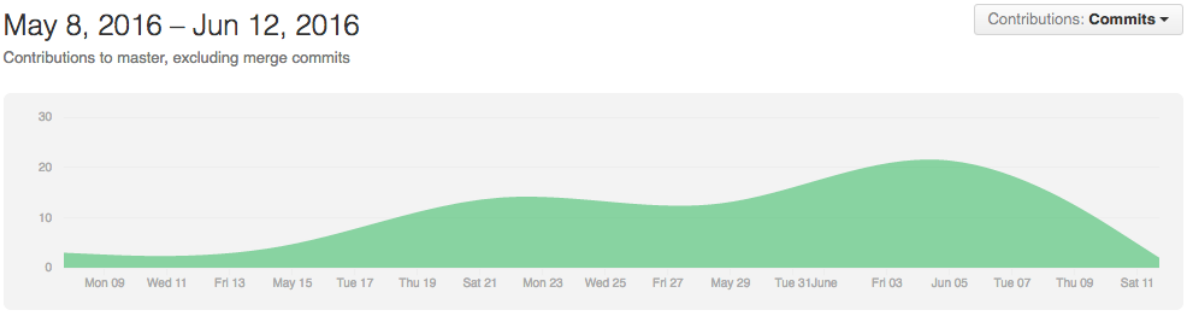
Slack : La réalisation de notre SMA nous demandant plus de temps que celui qui nous était proposé lors des séances de TD dédiées nous avons utilisé Slack, environnement de chat découpé en « channel » permettant, grâce à sa bonne intégration de Trello de pouvoir coupler les outils améliorant notre productivité. Ainsi, aucun autre médium de communication n'a été utilisé à par les réunions physiques et les « discussions slack. »



Chat Slack

Git : Un travail collaboratif sur un projet informatique conséquent nécessite, automatiquement, l'utilisation de l'outil de versioning « Git ». Nous avons pu scinder le développement et mener, en parallèle, les différents volets de développement. Ce fut l'occasion pour le dernier membre du groupe de s'approprier cet efficace outil pour les informaticiens.

Github : Afin d'héberger notre code, nous avons utilisé les repositories privés proposés par le pack étudiant de GitHub. Ainsi nous pouvions travailler de manière collaborative sur un projet tout au long de ces semaines de développements tout en gardant notre développement restreint au membre du groupe et non pas en public.

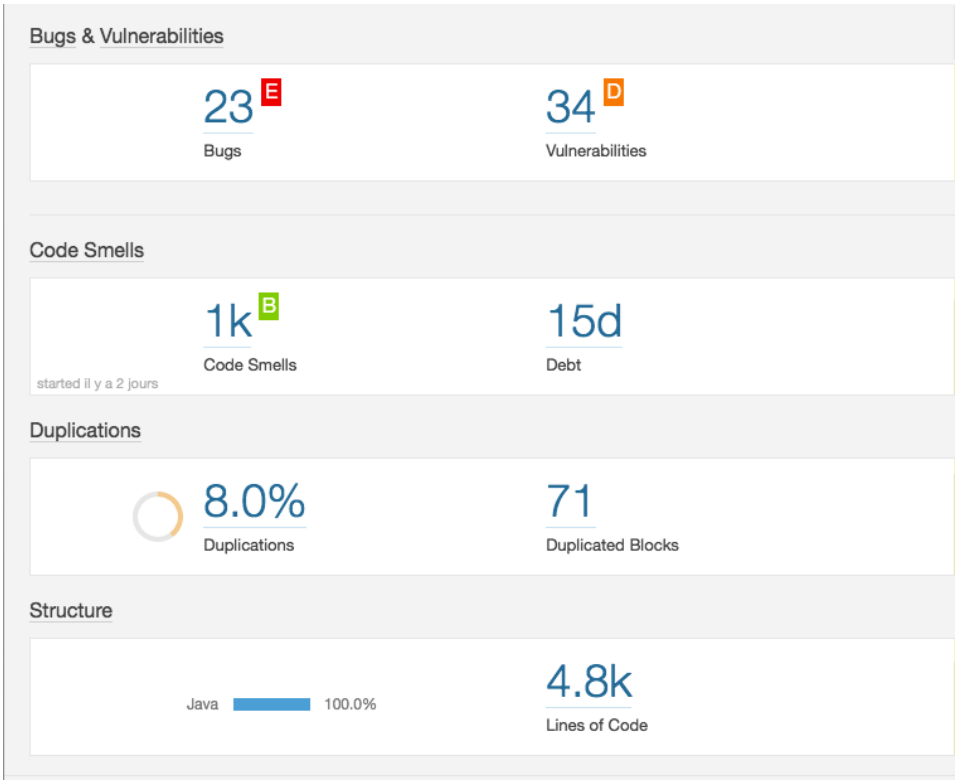


Graph des commit

JavaDoc : La décision fut rapidement prise de proposer une documentation Java accompagnant notre code. En effet, un SMA nécessite rapidement des centaines de lignes pour réaliser un projet conséquent comme le nôtre. Afin de faciliter la compréhension, à la fois par les membres du groupe, que par les extérieurs nous avons pris le temps de commenter notre code dans le but de fournir cette documentation. Un SMA se prêtant d'autant plus à cette pratique que le code JAVA qui en découle est très bien structuré.

Note : Vous trouverez ci-joint, dans le dossier « JavaDoc », la documentation au format HTML de notre SMA.

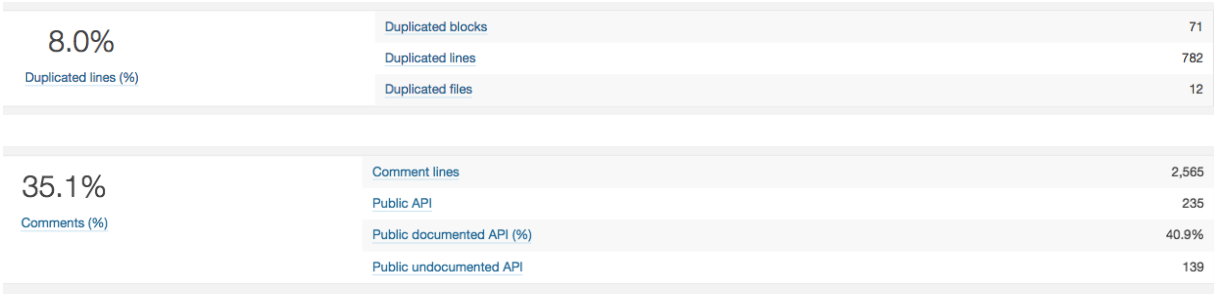
SonarQube : Ayant deux membres déjà sensibilisé aux problématiques d'optimisation de code ainsi qu'aux outils disponibles pour une analyse fine et profonde de milliers de lignes, nous avons décidé de travailler, lors de la dernière moitié du projet, avec SonarQube. Ainsi nous avons pu compléter les indications données par le compilateur par des recommandations plus précises (avec leur justification) venant de l'analyse de SonarQube. Même si l'utilisation de ce logiciel d'évaluation de la qualité du code produit n'était absolument pas une nécessite lors de ce projet d'IA04, nous avons été pris par la dynamique de notre développement et c'est tout naturellement qu'un projet aussi important nous a permis de tester de nombreux outils qui était insignifiants sur des productions moins conséquentes.



Grandeurs principales mesurées par SonarQube

Note : Nos scores ne sont pas très bons concernant les bug et les vulnérabilités. Cela s'explique par une utilisation massive des affichages dans la console qui ne sont pas recommandés par SonarQube, qui préfère l'inscription dans un fichier de log. (tout comme les exceptions venant des interruptions des try-catch.

Indication de la maintenabilité de notre code



Grandeurs complémentaires sur la documentation de notre code ainsi que sur sa structure

4. Difficultés rencontrées

Spécifiquement à KB :

- L'agent KB nous a posé quelques soucis. En effet nous avions créé nos cinquante lois dans notre ABOX. Cependant s'est rapidement posé le problème de l'exploitation de cette base. Nous voulions faire une requête sur cette base pour en récupérer les lois correspondant à un parti et qui n'ont pas été encore votées. Le problème est venu de cette seconde contrainte : « non encore votée ». En effet cela signifiait que l'on devait pouvoir modifier notre modèle au fur et à mesure de l'évolution de notre jeu (au fur et à mesure que des lois étaient votées par le parlement). Nous devions donc pouvoir requête et mettre à jour notre modèle.
  Nous avons longtemps hésité entre deux techniques différentes. Soit en dupliquant notre base de lois dans un second fichier que l'on éditerait selon l'évolution du jeu. Soit nous tirions parti de JENA qui nous offrait la possibilité de manipuler un modèle RDF issu de l'importation de notre ABOX.
  Pour des soucis de faciliter d'implémentation nous avons choisi la seconde solution.

- Le prise en main de ce modèle JENA basé sur des « statement » et des « resource » fut un autre problème. JENA stocke le modèle selon ses propres conventions (en rajoutant des informations sur l'URI dans les chaines stockées, on ne proposant que très difficilement la gestion des entiers). Nous avons donc décidé de refaçonner notre base de lois en changeant le type de notre objet pour tous les passer comme chaine de caractères. Grâce à quelques iterator sur des listes de statement nous avons pu mettre en place de nombreuses fonctions

de manipulations du modèle JENA. Grâce à eux nous avons facilement pu requêter notre base pour récupérer les informations des lois et aussi mettre à jour la propriété « is_voted » à chaque fois qu'une loi était votée.

De manière plus générale :

- Notre système multi-agent repose sur un nombre important de messages envoyés par de nombreux agents différents. Afin de ne pas « parasiter » les schémas de communications propre à chaque action nous avons dû mener une réflexion plus importante que pour les précédents projets concernant le choix des performatifs.

- La mise en place des schémas de communication entre nos différents agents (use-case) pour chaque action à réaliser a fait l'objet de nombreuses réflexions au sein du groupe. En effet nous avons décidé d'implémenter le plus d'agents possibles pour nos besoins afin de découpler les tâches et éviter de centraliser des actions totalement différentes. La contrepartie fut de poser des contraintes sur les schémas de communications. Les rôles respectifs des agents « utilisateur » et « médiateur » ont fait l'objet de nombreuses discussions

- La conception de l'intelligence artificielle en elle-même fut très compliquée. De nombreuses questions se sont posées à nous durant les phases de réalisation du projet : comment choisir les intervalles de nos variables ? Comment savoir si un député va voter ou non une loi ? Comment pondérer les paramètres à mettre en avant concernant le vote d'une loi ? Comment équilibrer le jeu afin qu'il ne tende pas trop rapidement vers un extrême ou l'autre (victoire ou défait) ? Comment modéliser l'hésitation d'un député au moment d'un vote ?

- La fusion du code fût problématique pour l'action de l'agent Rumeur. Puisque que ce-dernier échange plusieurs messages avec l'agent Utilisateur durant l'action de "Répandre une rumeur", il est nécessaire que l'agent Utilisateur soit toujours actif, ce qui n'est pas toujours le cas : suite à l'ajout de l'interface graphique, cet agent, plus précisément son "thread", se retrouve bloqué lorsqu'une fenêtre attend qu'un bouton soit appuyé. Par conséquent, il n'est pas capable de répondre aux requêtes venant de l'agent Rumeur et, ainsi, l'action se retrouve bloquée. Il a fallu changer le fonctionnement de l'agent Utilisateur et celui de

5. Idées d'amélioration

Concernant le jeu :

- Afin de créer un jeu complet et jouable dans une partie « long-terme » il serait intéressant d'avoir une base de lois de milliers de lois différentes.

- Pour éviter que le jeu soit répétitif d'une partie à l'autre nous pourrions, au démarrage d'une partie sélectionner seulement certaines lois de la base et ainsi ne jamais (ou presque) faire de partie avec les mêmes lois à voter ou proposer.

- Rajouter des niveaux de difficulté au démarrage du jeu en rendant les députés adverses plus féroces ou bien en faisant démarrer le joueur d'un niveau plus faible.

- Implémenter encore plus d'actions pour une richesse de jeu plus importante. Proposer par exemple tous les 50 tours de modifier la composition du parlement (simulation d'élections).

- Implémenter un mode multi-joueurs, pour que plusieurs personnes puissent être en concurrence, ce qui ajoutera une nouvelle dimension au jeu (vu qu'un joueur humain sera très probablement plus performant que l'IA des députés).

- Permettre aux députés de choisir et faire des actions pour rendre le jeu encore plus dynamique. Pour l'instant, les députés ne peuvent que proposer et voter des lois.

- Implémenter des évènements aléatoires, ayant des conséquences positives ou négatives sur l'environnement ou bien sur les membres du parlement.

- Donner la possibilité de sauvegarder l'état du jeu pour que ce soit possible de reprendre le jeu à un autre moment. Ceci pourrait être fait en sérialisant les caractéristiques de chaque agent (sauf les variables contenant les Aïds connus, qui, elles, seront de nouveau récupérées depuis l'agent DF à la reprise du jeu).

Concernant le SMA :

- Changer les performatifs des messages échangés en les rendant plus appropriés aux situations pour lesquelles ils étaient utilisées.

- Rendre les agents plus indépendants pour que le système soit plus modulaire. Actuellement, les agents sont de plus en plus dépendants les uns des autres avec l'ajout de complexité au jeu. La réduction de la dépendance pourra permettre d'enrichir le jeu plus facilement dans l'avenir.

Conclusion

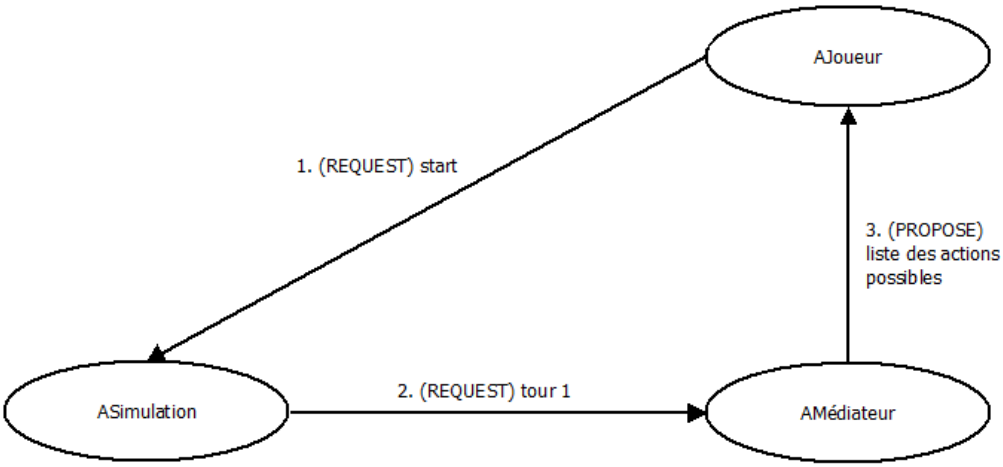
La conduite de ce projet d'envergure fut pour nous l'occasion de créer pour la première un système multi-agents selon nos propres envies. C'est comme une suite logique et gratifiante des cours et les TD que nous avons perçu la réalisation de ce travail.

Pendant les premières semaines de ce semestre nous avons pu nous confronter à la réalisation de système multi-agents tout en étant guidé que ce soit pour la conception ou la réalisation. Au fur et à mesure que les semaines passées nous avant acquis l'expérience et les réflexes pour finalement mener à bien un projet important.

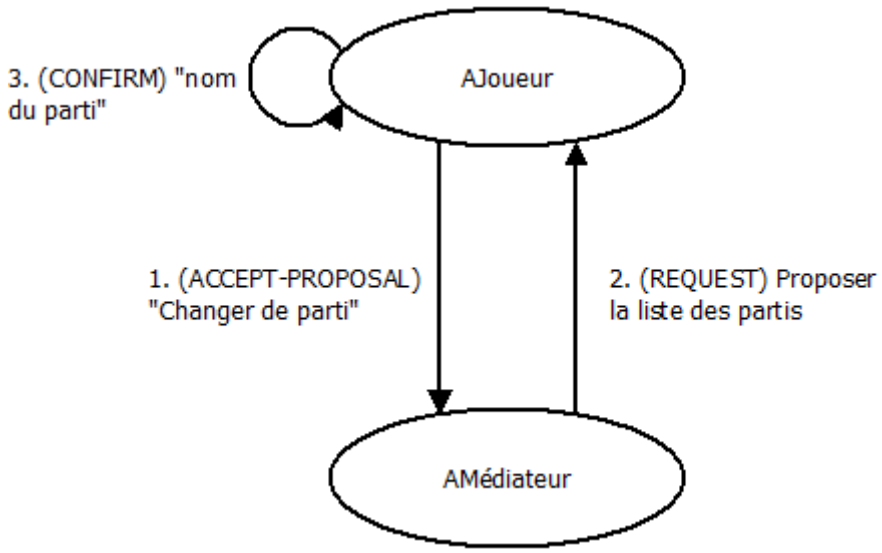
Nous étions tous néophyte face au système multi-agents avant de suivre le premier cours d'IA04 et nous voici, en concluant ce rapport en phase de finalisation d'un projet issu de nos envies personnelles.

Ce projet constituait à son lancement un challenge pour nous devant l'apparente complexité de celui-ci et c'est satisfait que nous sortons de sa réalisation. Satisfait d'avoir réussi à modéliser, non pas parfaitement, mais d'une manière qui nous satisfasse un comportement propre à l'homme : le vote.

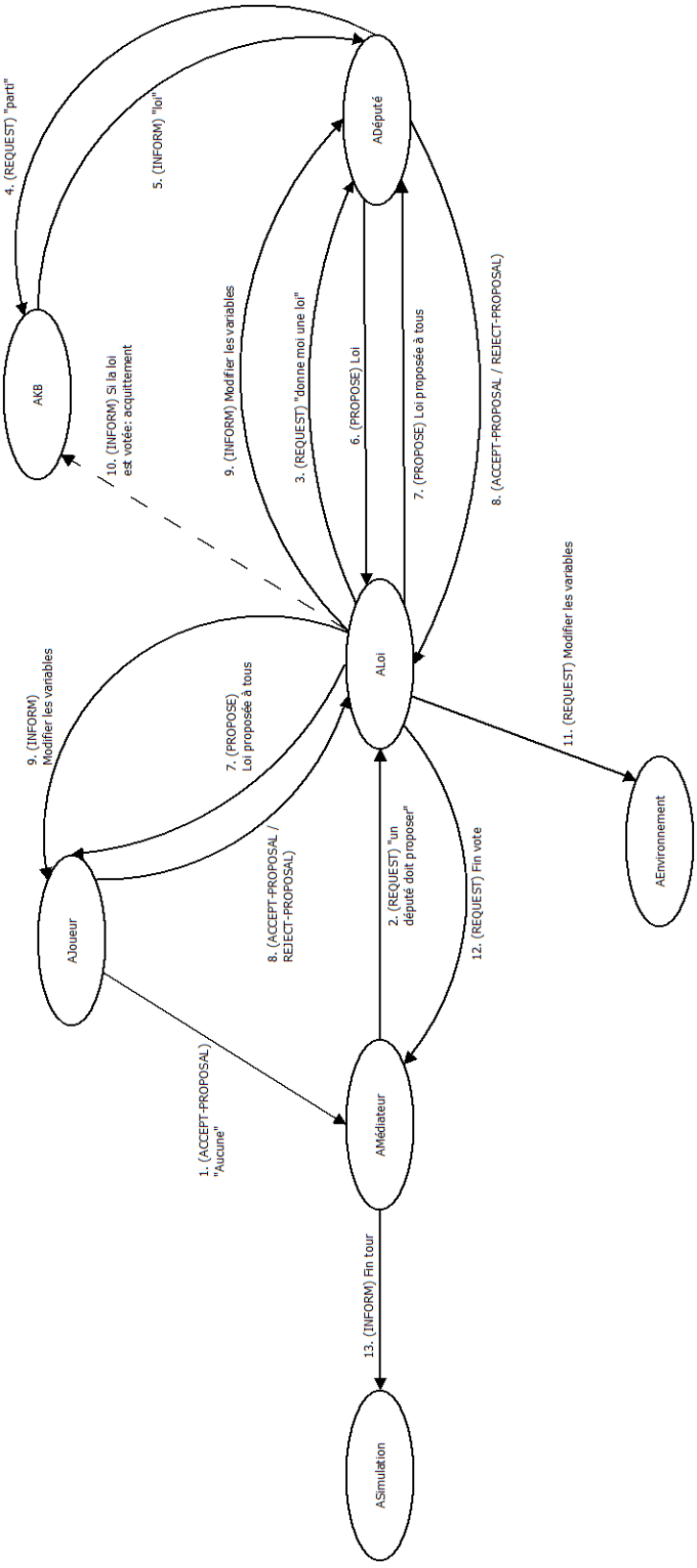
Annexes



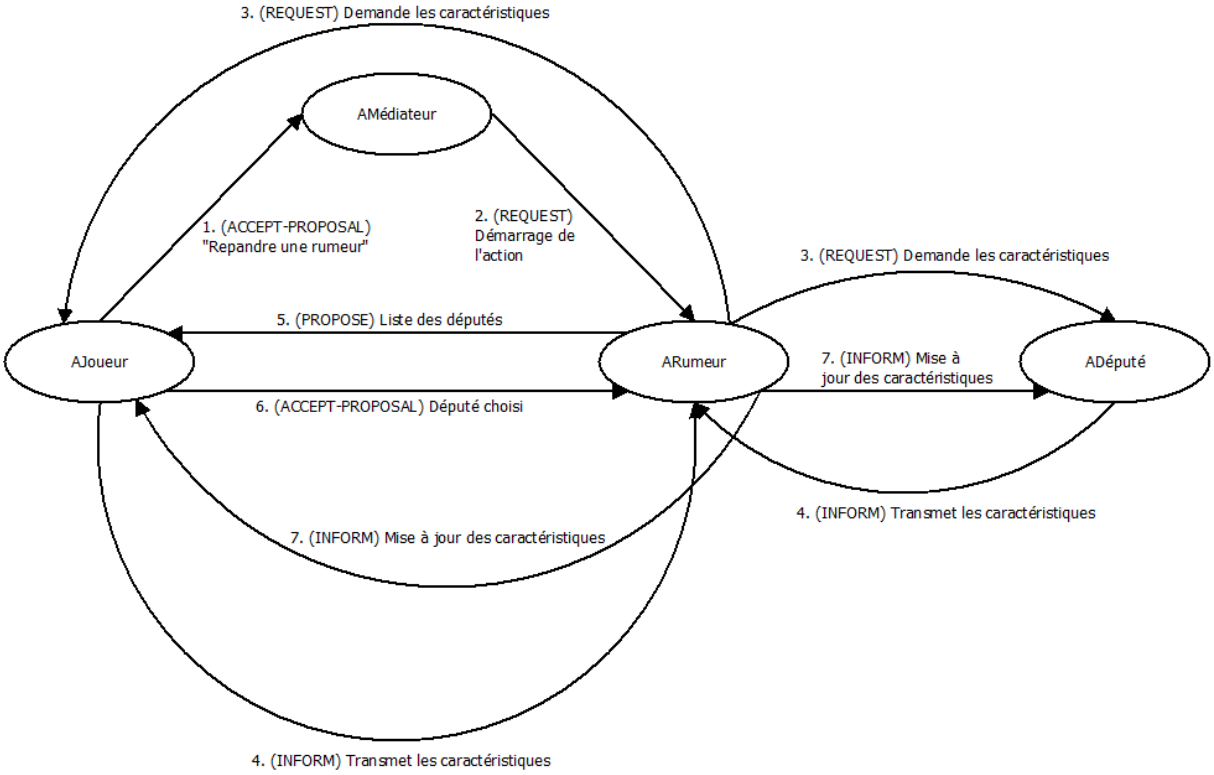
Messages échangés au démarrage du système



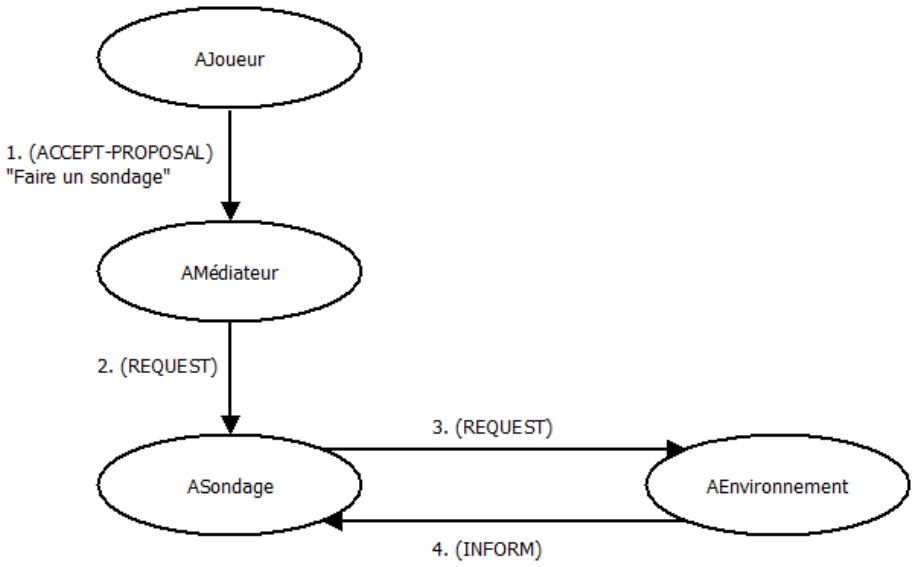
Messages échangés lors de l'action "Changer de parti"


ADéputé
AKB
ALoi
AJoueur
AMédiateur
ASimulation
AEnvironnement
4. (REQUEST) "parti"
5. (INFORM) "loi"
10. (INFORM) Si la loi est votée: acquittement
9. (INFORM) Modifier les variables
3. (REQUEST) "donne moi une loi"
6. (PROPOSE) Loi
7. (PROPOSE) Loi proposée à tous
8. (ACCEPT-PROPOSAL / REJECT-PROPOSAL)
9. (INFORM) Modifier les variables
7. (PROPOSE) Loi proposée à tous
8. (ACCEPT-PROPOSAL / REJECT-PROPOSAL)
2. (REQUEST) "un député doit proposer"
12. (REQUEST) Fin vote
11. (REQUEST) Modifier les variables
1. (ACCEPT-PROPOSAL) "Aucune"
13. (INFORM) Fin tour

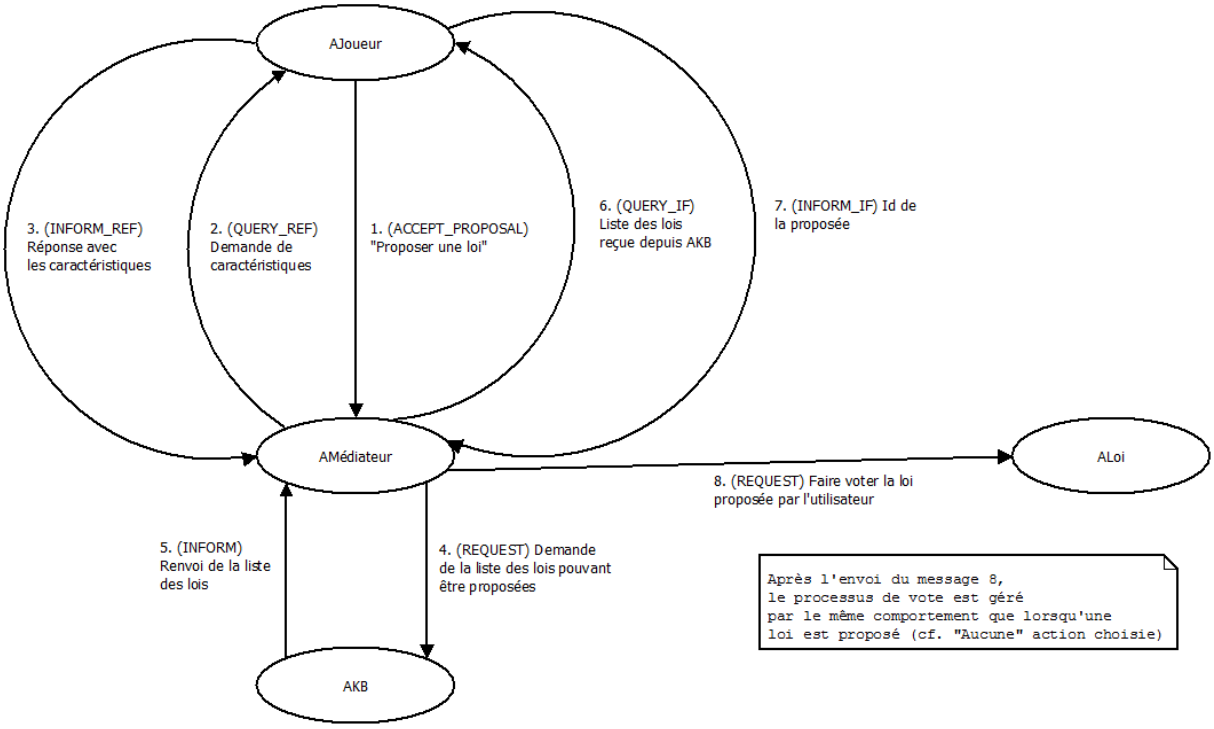
Messages échangés lors de l'action "Aucune" (simple vote d'une loi proposée par un autre député)



Messages échangés lors de l'action "Répandre des rumeurs"



Messages échangés lors de l'action "Sondage de l'environnement"



Messages échangés lors de l'action "Proposer une loi"