Projet ACVL/CAWEB

Jeu du Loup-Garou

11 avril 2017

Maxime Deloche, Ludovic Carré & Cyril Carlin

Table des matières

1 Analyse

1.1 Diagrammes de cas d'utilisations

1.1.1 En partie

Ces diagrammes représentent les cas d'utilisation possibles de l'application, lorsque l'utilisateur est dans une partie. Nous avons distingué 2 cas (pendant le jour et pendant la nuit) pour améliorer la lisibilité, les actions possibles étant très différentes entre les 2 phases.

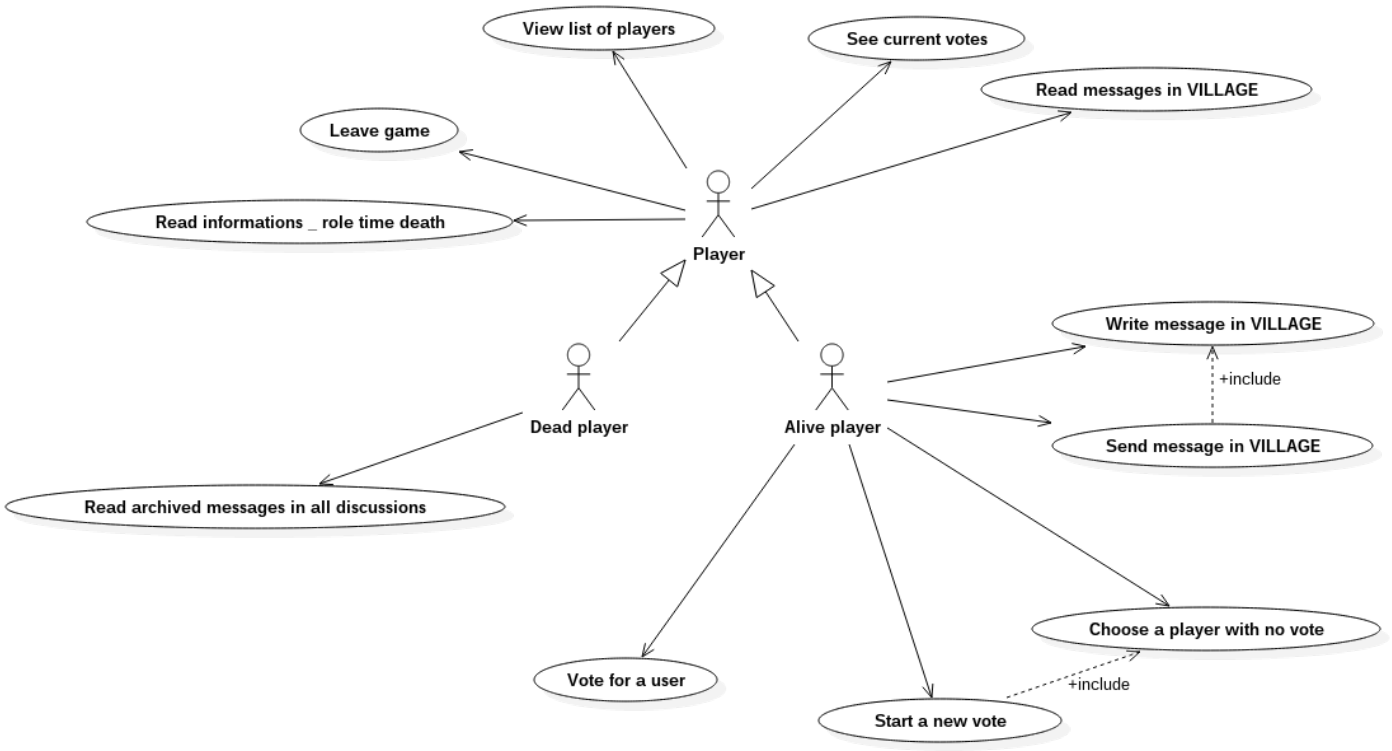

FIGURE 1 – In Game (during day) use case diagram

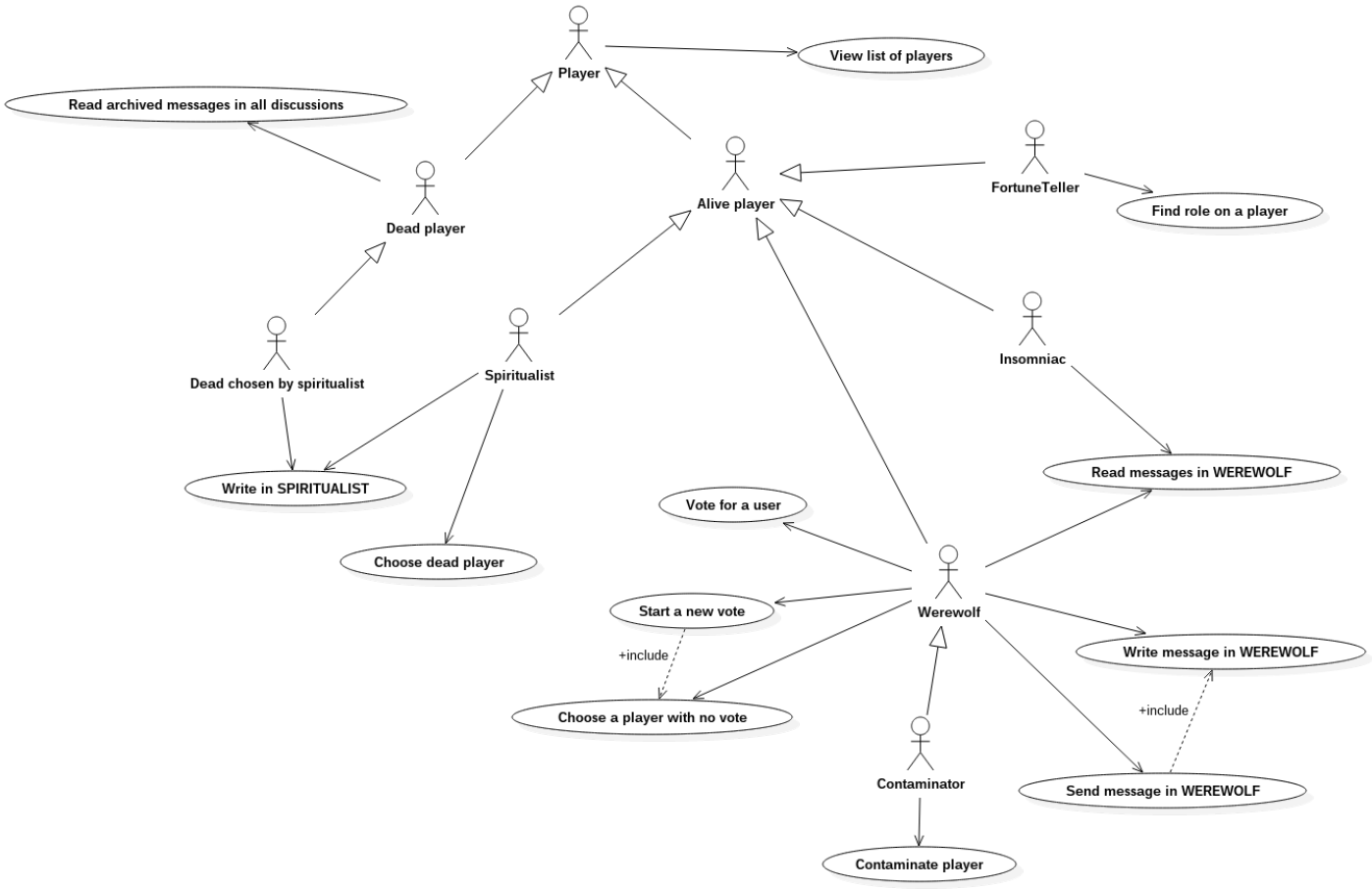

FIGURE 2 – In Game (during night) use case diagram

1.1.2 Hors partie

Ce diagramme représente les cas d'utilisation lorsque l'utilisateur n'est pas dans une partie.

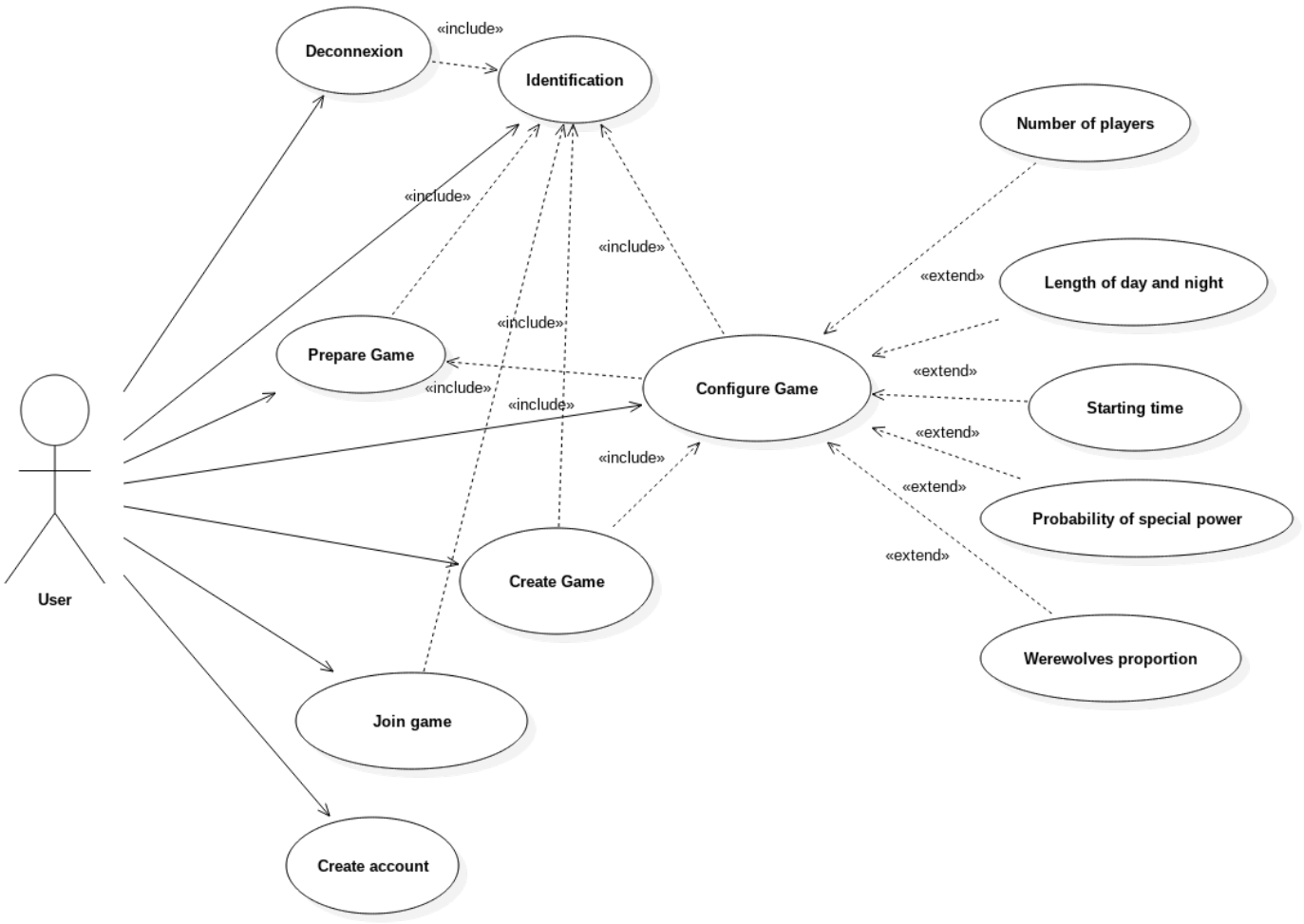

FIGURE 3 – Out of game use case diagram

1.2 Diagrammes de classes d'analyse

Ce diagramme représente l'analyse des différentes classes nécessaires à la modélisation de l'application.

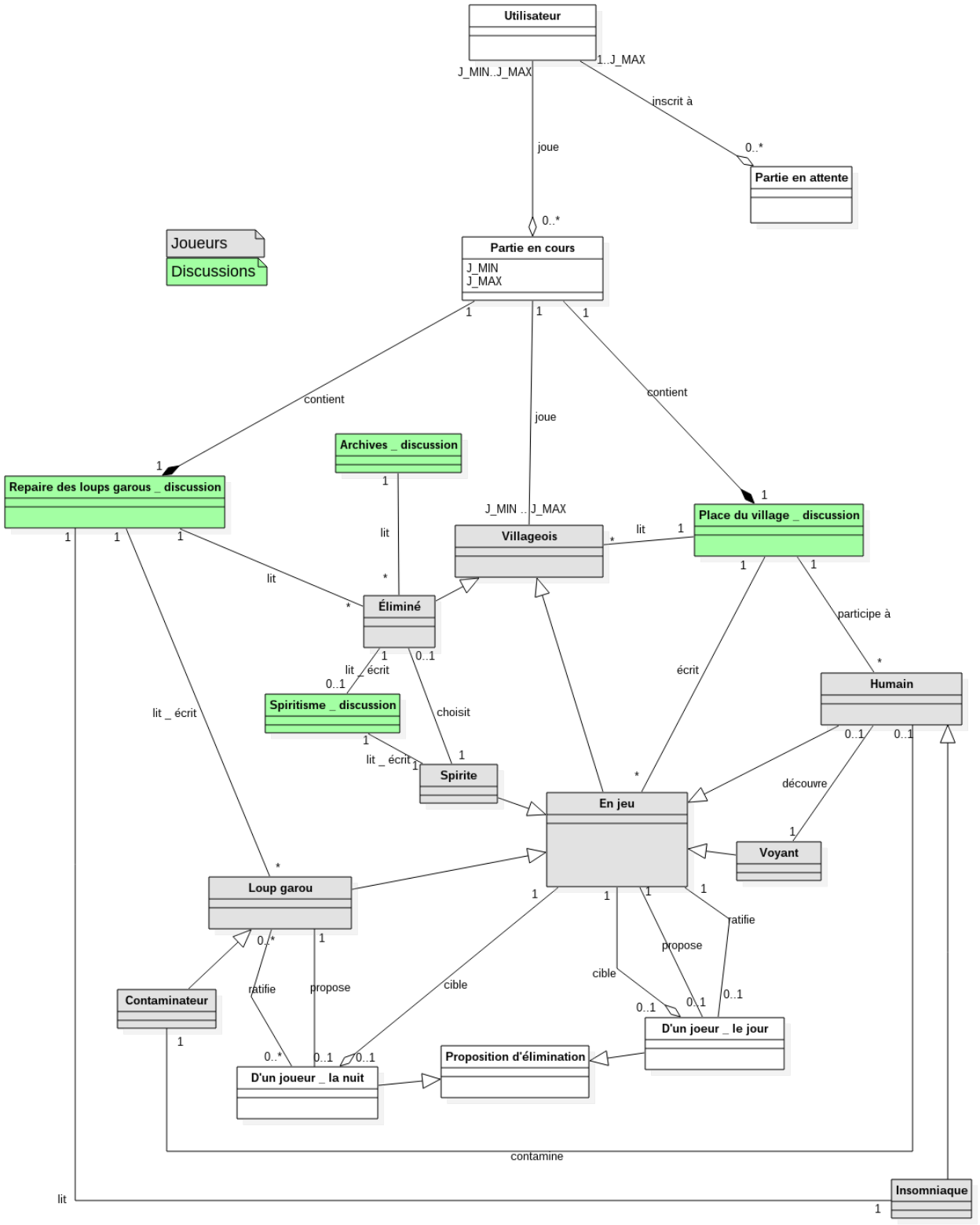

FIGURE 4 – Class analysis diagram

Note : à cette étape du projet, on supposait que les pouvoirs spéciaux étaient pour la plupart accessibles aux villageois et aux loups-garous. Nous avons par la suite discuté ce point, voir la partie "Bilan - interprétations du sujet".

1.3 Diagrammes de séquence système

1.3.1 Voting system

Ce diagramme représente la séquence de communications lorsque l'utilisateur choisir d'initier un nouveau vote. Comme le texte l'explique ci-dessous, toutes les communications entre client et serveur se font selon le même modèle.

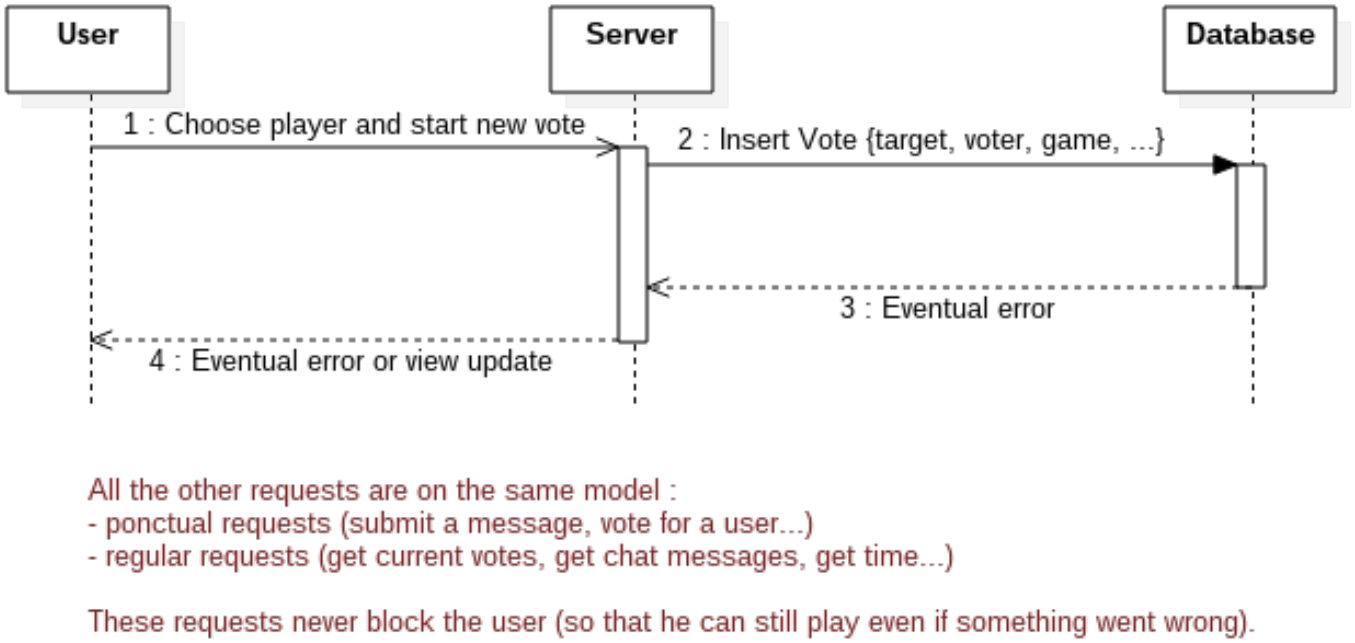

FIGURE 5 – In game sequence diagram

1.3.2 Gestion du temps

Ce diagramme de séquence représente la création de threads au démarrage du serveur Tomcat ainsi que leur destruction lorsqu'il est éteint. Ces threads sont utilisés pour gérer l'écoulement du temps au cours de la partie ainsi que la fin de la partie lorsqu'une équipe à gagné.

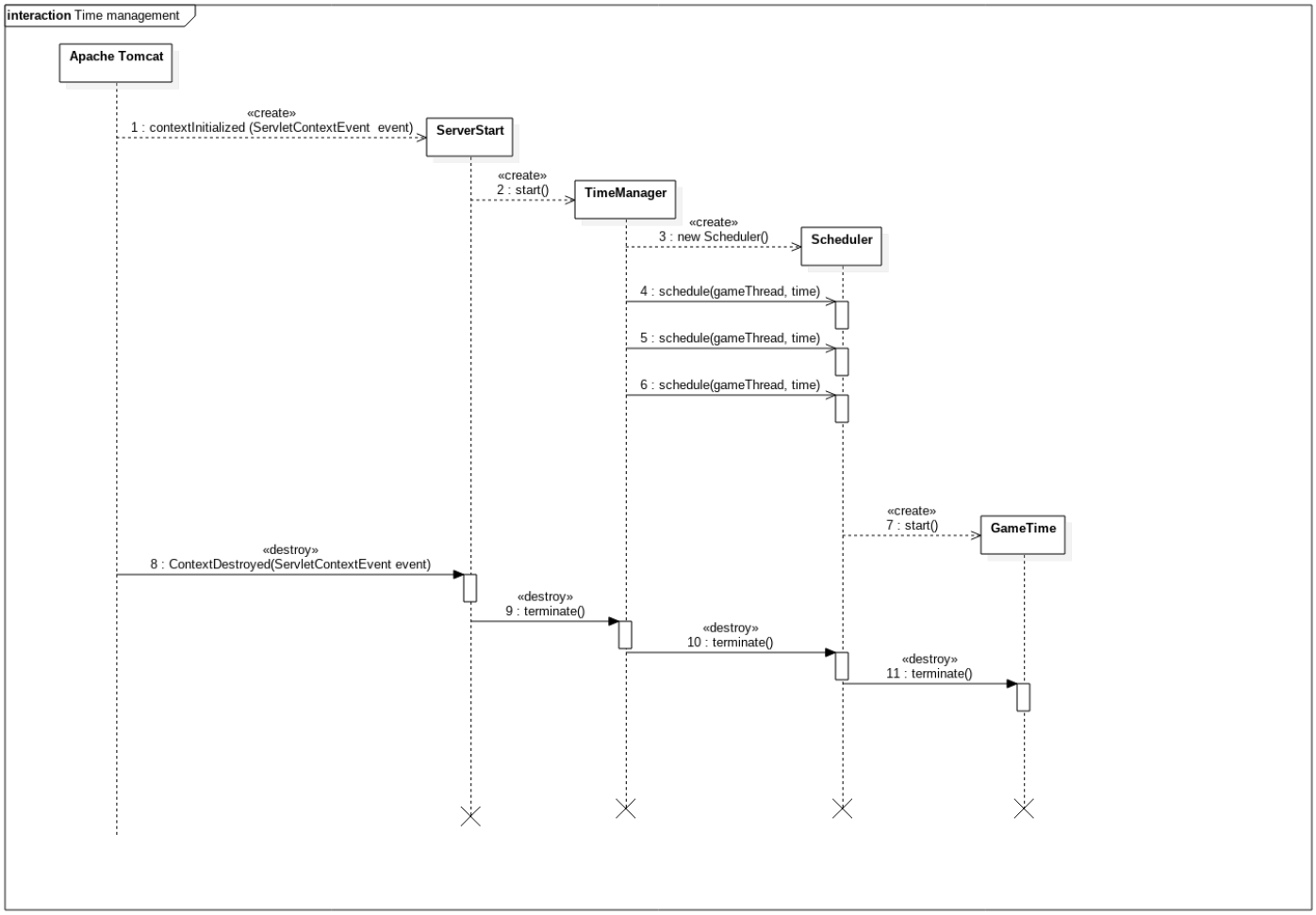

FIGURE 6 – Thread factory diagram

1.4 Diagrammes d'activité

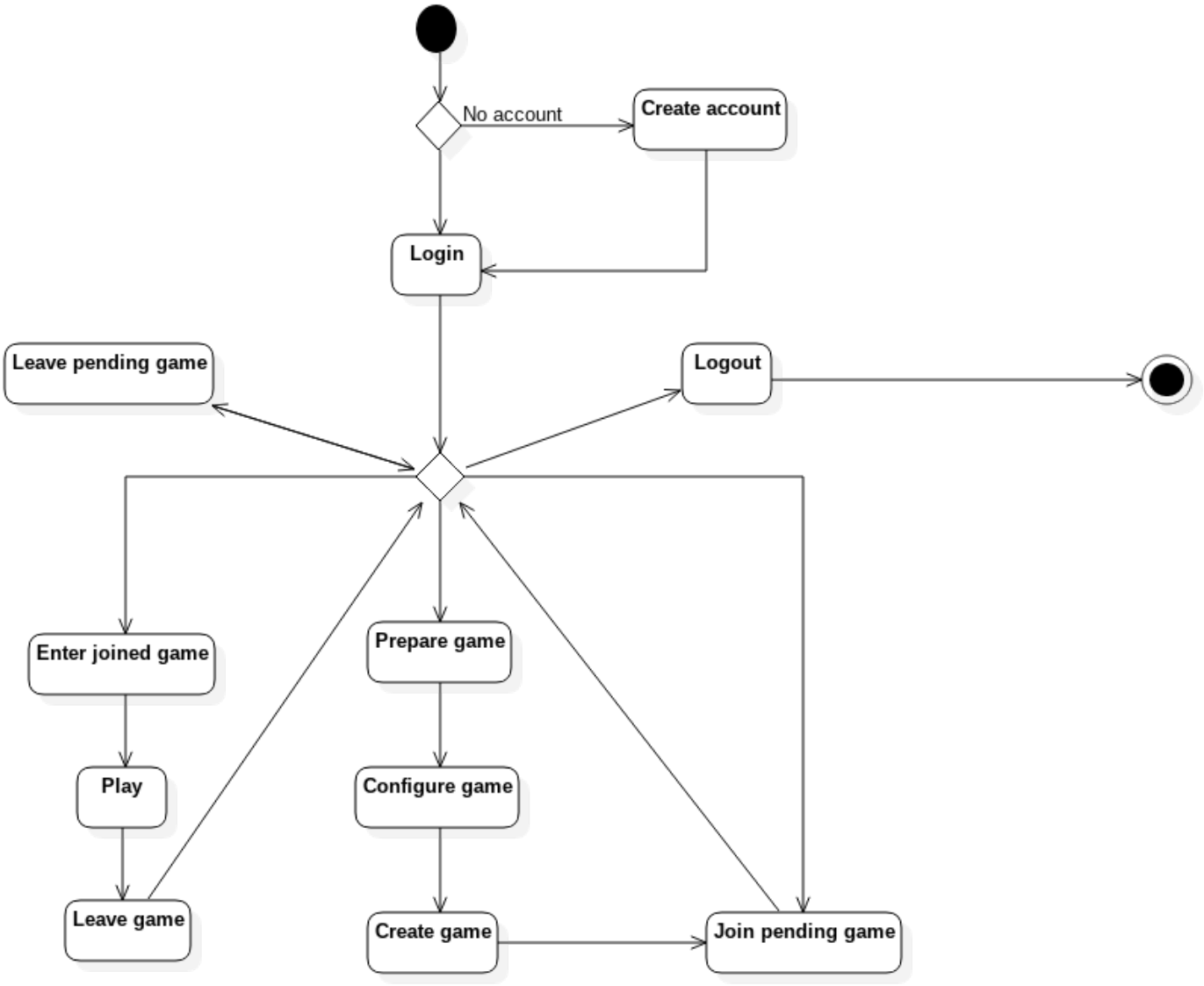

FIGURE 7 – Out of game activity diagram

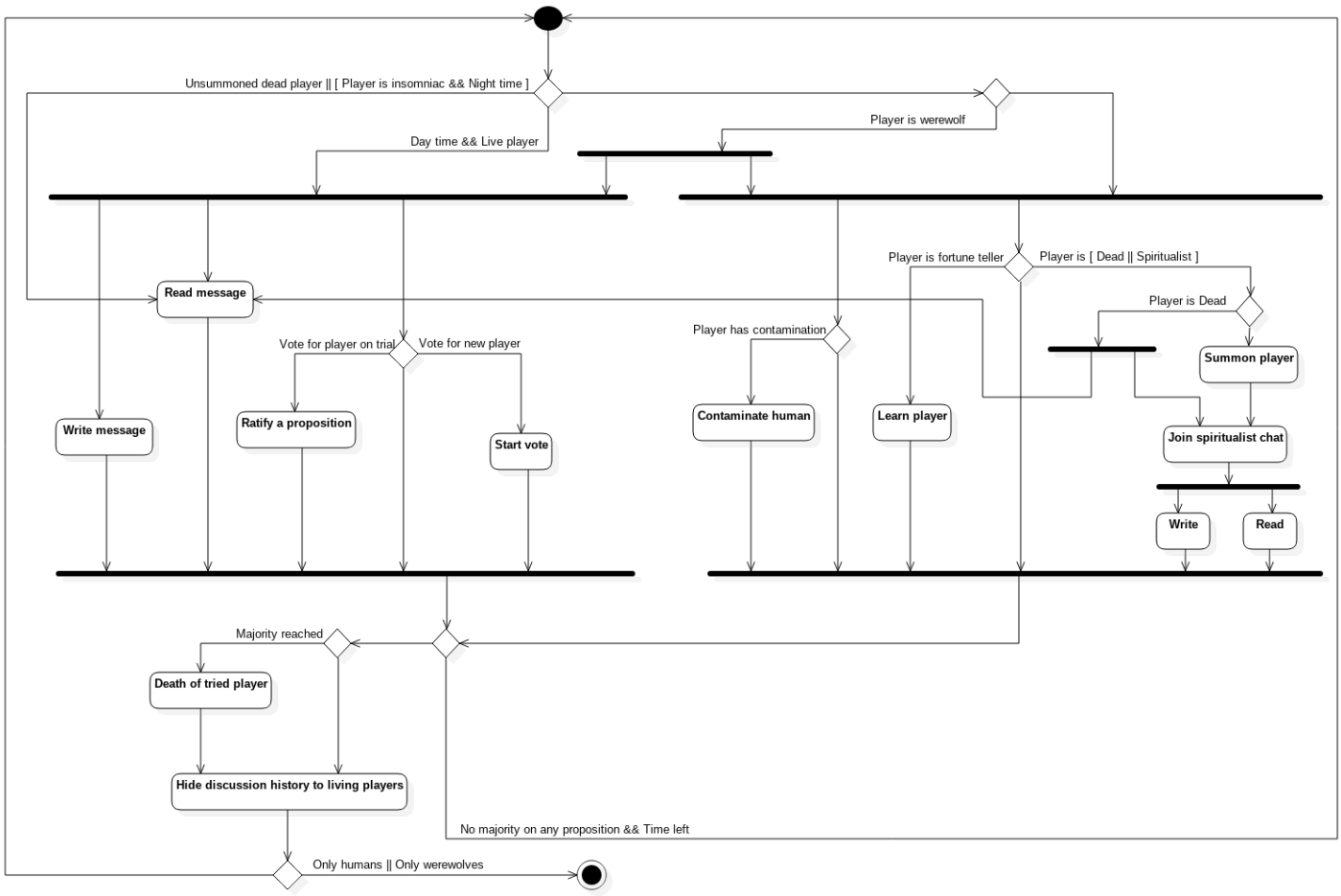

FIGURE 8 – In game activity diagram

2 Conception

2.1 Architecture MVC

La figure ci-dessous montre comment l'architecture MVC est implémentée dans notre projet. Tous les liens entre les classes ne sont pas représentés par souci de lisibilité.

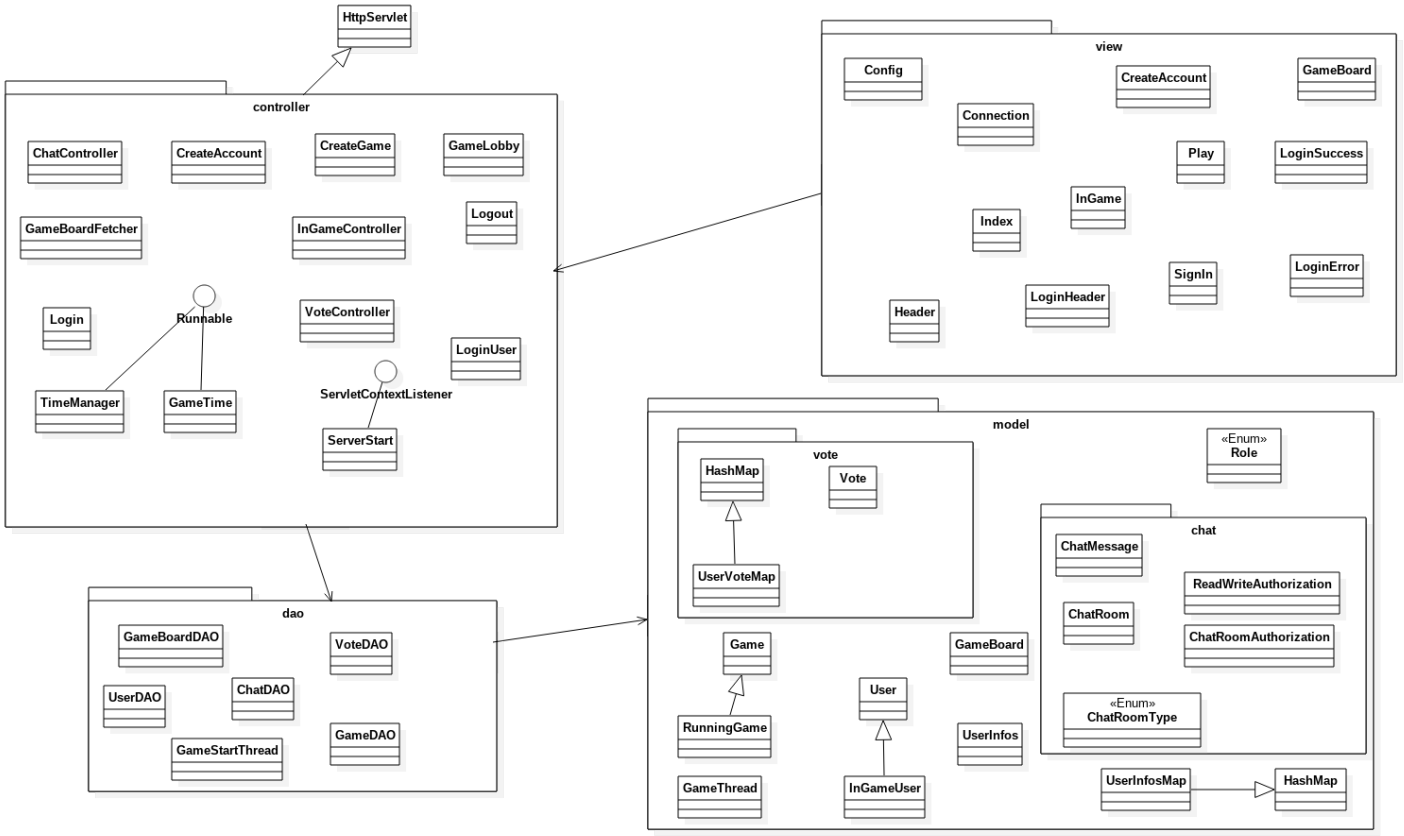

FIGURE 9 – Out of game activity diagram

2.2 Conception de la base de données

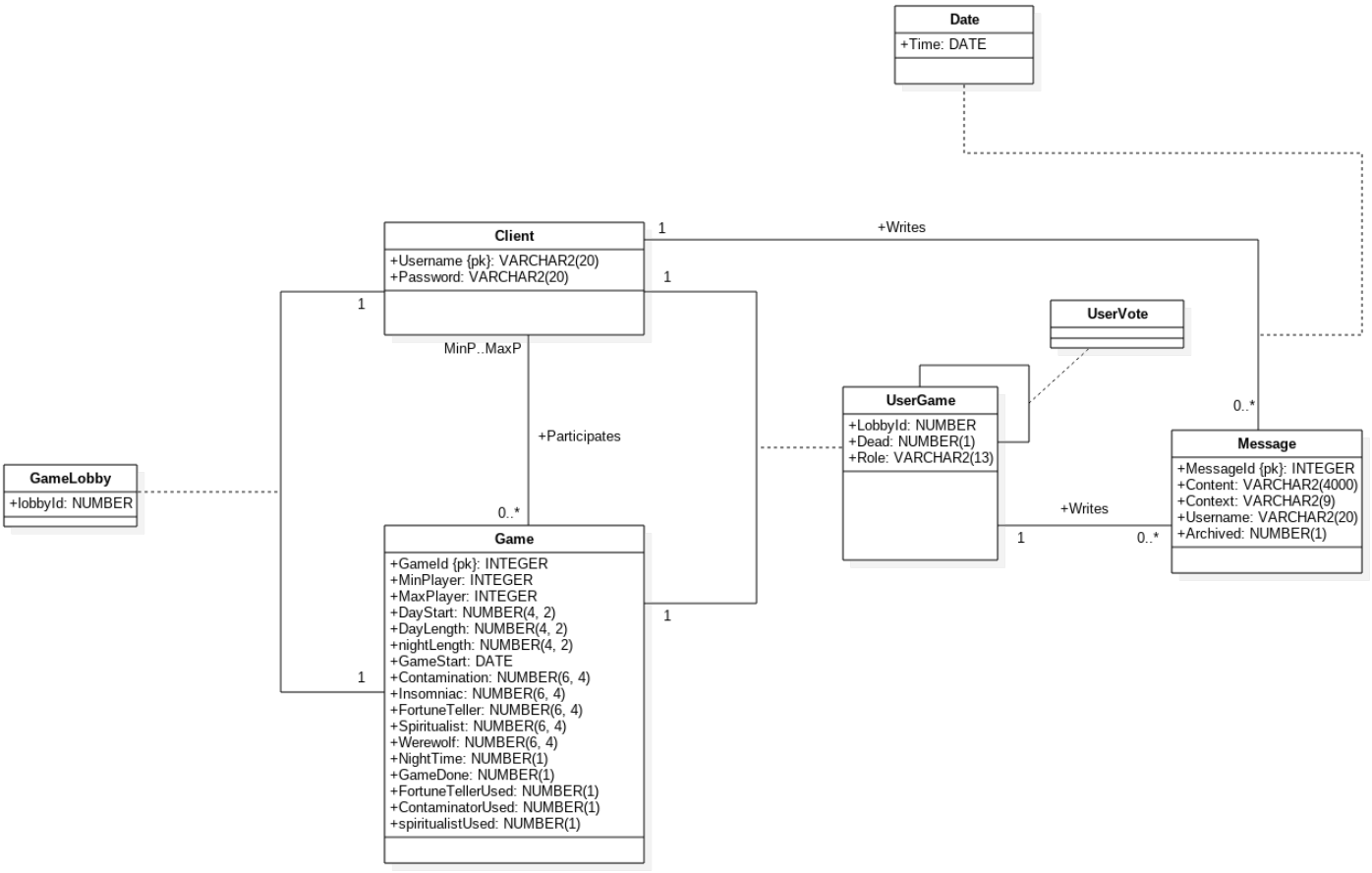

FIGURE 10 – Database conception diagram

2.3 Diagrammes de classes de conception

Ci-dessous, on trouve le détail de la partie Model de l'architecture MVC.

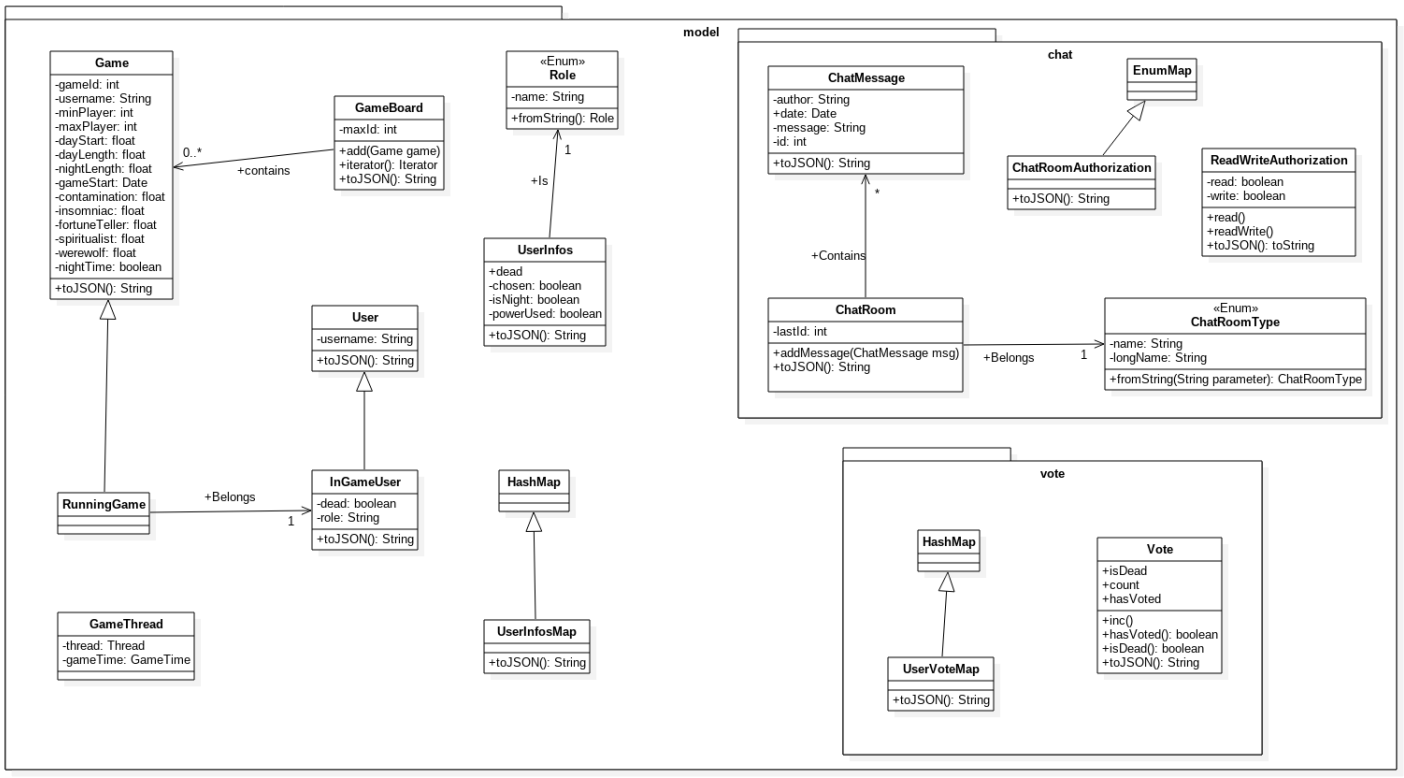

FIGURE 11 – Conception diagram of the model part

2.4 Diagrammes d'états-transitions

Ce diagramme représente les transitions entre les différents états possible s de l'utilisateur (en jeu et hors-jeu).

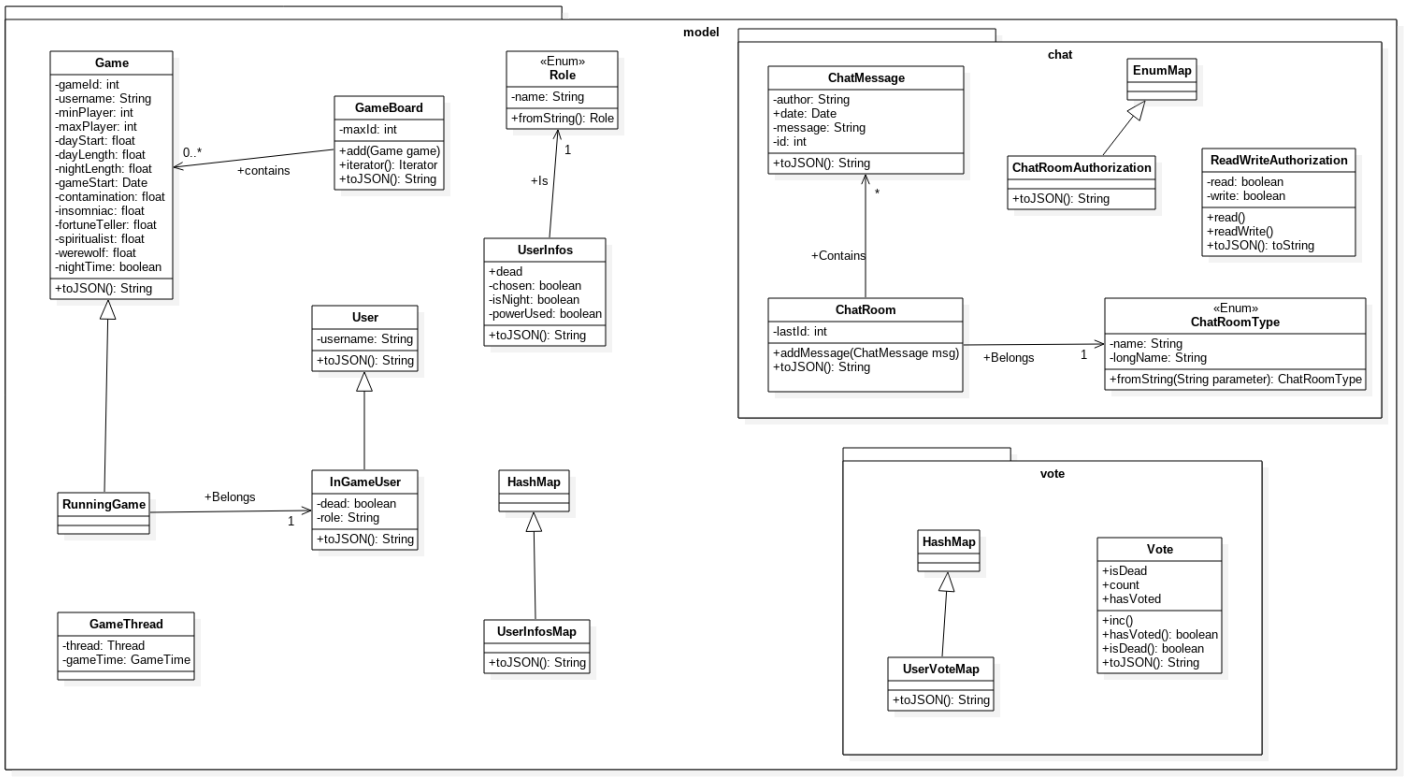

FIGURE 12 – Conception diagram of the model part

3 Manuel utilisateur

3.1 Hors partie

Quand l'utilisateur n'est pas en partie, il peut naviguer à travers 4 pages : index.html, play.html, config.html et connection.html. Ces 4 pages sont accessibles à partir de la barre de navigation qui est récurrente dans chaque page et située en haut.

La page d'accueil est une page dédiée uniquement à la présentation brève et rapide du site, et peut accueillir une extension qui est la barre de recherche pour pouvoir accéder à un type de parties précis ou même accéder à une partie rapidement. Cet accueil est accessible depuis l'onglet "HOME" de la barre de navigation.

Le bouton "PLAY" permet d'afficher la liste de toutes les parties en attente. Le joueur peut alors s'inscrire à une partie, et cette partie est ajoutée à son gameboard.

L'onglet "GAMEBOARD" affiche la liste des parties propres à l'utilisateur. Il peut voir les paramètres d'une partie en attente, et entrer dans le jeu d'une partie en cours.

Enfin, le bouton "CREATE" permet de commencer la configuration d'une nouvelle partie.

Lorsque l'utilisateur n'est pas connecté, on trouve un bouton "Sign in" qui permet au joueur de se connecter ou de créer un nouveau compte. Une action qui nécessite d'être identifié redirigera également sur cette page. Lorsque l'utilisateur est connecté, on trouve à la place un bouton "Sign out" permettant de se déconnecter et de revenir à la page d'accueil.

3.2 Configuration de partie

La configuration d'une partie se fait de manière très simple sur la page config.html, accessible depuis l'onglet "CREATE" dans la barre de navigation. Une fois les paramètres saisis et la partie créée, elle est ajoutée au gameboard du joueur. La saisie des paramètres est controllée et contrainte. Par exemple, le format de la date suit la règle 'yyyy-mm-dd' ; les paramètres de pouvoirs spéciaux ne

3.3 Gameboard

La page Gameboard est une page accessible uniquement après connection de l'utilisateur. Elle lui présente une liste de parties qui le concernent en commançant par les parties qui n'ont pas encore commencé et qu'il a rejoint ; puis les parties déjà commencées et qu'il a également rejoint au préalable. Ces dernières peuvent être reprises à partir du bouton "Resume" situé dans l'encart de la partie concernée.

3.4 En partie

L'interface en jeu est constituée de 4 parties :

— A gauche se trouve le ou les chats disponibles pour le joueur. Les nouveaux messages s'affichent automatiquement, et l'utilisateur peut changer de chat (parmi ceux auxquels il a accès, en fonction de son rôle et de la période) via le menu déroulant au dessus, et envoyer un message via le formulaire en-dessous. Notez qu'à la fin de la partie, tous les messages sont publiés.

— Au centre se trouve la partie consacrée aux votes. Le joueur voit, pour la période courante, les joueurs déjà proposés au vote ainsi que le nombre de votes pour ce joueur. Il peut, si il ne l'a pas déjà fait, voter à son tour pour ce joueur (via le bouton sur la droite, qui est grisé après avoir voté). Enfin, l'utilisateur peut soumettre un nouveau vote via le formulaire en-dessous (qui ne contient que les joueurs n'ayant pas de vote). Notez que les fonctionnalités de vote sont inaccessibles lorsque l'utilisateur est mort, ne peut pas voter (un humain la nuit), ou lorqu'un vote a atteint une majorité.

— A droite se trouve la liste des joueurs de la partie, et indique si ils sont éliminés.

— Le bandeau en bas de l'écran contient les informations sur l'utilisateur : son nom, son rôle, si il est éliminé... C'est également ici qu'apparait l'interface pour les pouvoirs spéciaux. Si le joueur est spiritualiste, contaminateur ou voyant, un menu déroulant avec une liste de joueurs pouvant être sélectionnés s'affiche. Le joueur peut alors utiliser son pouvoir sur le joueur de son choix, à la suite de quoi cette zone est désactivée.

3.5 Fin de partie

A chaque fois qu'un vote est validé, une alerte apparait sur tous les écrans de joueurs connectés à la partie, indiquant qui est éliminé (mais pas son rôle, voir la partie "Interprétations"). Si ce vote était décisif, c'est-à-dire qu'il ne reste plus que des joueurs d'un même camp, ce camp gagne. Une alerte est alors affichée, indiquant l'équipe gagnante, et les discussions archivées sont publiées.

4 Bilan

4.1 Choix d'implémentation

4.1.1 Gestion de l'affichage dynamique

La page inGame est affichée dynamiquement, et n'a jamais besoin d'être rechargée. Nous utilisons des requêtes jQuery, qui interrogent en continu des servlets. Ces servlets interrogent la base de données via un DAO et renvoient les informations requises dans le format JSON. Celui-ci est ensuite parsé par un handler associé à la requête jQuery, et le contenu de la page (les messages, les votes, le statut de joueurs, la période...) est actualisé dynamiquement via du JavaScript.

Nous avons choisi d'utiliser une page dynamique car il ne nous paraissait ni ergonomique ni intuitif pour l'utilisateur de devoir recharger manuellement la page pour recevoir les nouveaux messages, votes...

Le code de cette partie est visible dans le fichier `in_game.jsp` et dans les fichiers `user.js, vote.js et chat`

4.1.2 Gestion du temps : *threads*

Le cahier des charges indique que les parties doivent démarrer à l'heure indiquée et que chaque période *nuit/jour* a une durée donnée. Il est impossible de respecter cette demande du cahier des charges en effectuant les requêtes nécessaires au lancement de partie et au changement de période du côté du client en **javascript**. La validation de cette demande dépendrait du fait qu'il y ait toujours au moins un client sur la page du site à l'heure de lancement ou à l'heure de changement, ce que nous ne pouvons pas garantir.

Nous avons donc décidé d'utiliser un système de thread pour garantir une implémentation correcte de cette gestion du temps puisque le serveur est lui toujours allumé. La procédure est la suivante :

1. Le serveur est allumé (*startup.sh*), la méthode **ServerStart.contextInitialized** est alors appelé automatiquement.
2. Cette méthode créé un nouveau thread *TimeManager*.
3. Le **TimeManager** va à son tour parcourir la liste des parties qui ne sont pas encore lancées et si leur temps de lancement est supérieur au temps courant, il va prévoir la création d'un thread **GameTime** à la date de lancement indiquée à l'aide d'un scheduler.
4. Lorsque la date de lancement d'une partie est venue, le scheduler va lancer le thread associé à la partie.
5. Ce thread va alors vérifier si le nombre de joueurs minimum est atteint et dans ce cas, il se met en attente de la fin de la période courante.
6. A chaque fin de période, le thread vérifie si la partie est finie ou non. Si elle l'est, il l'indique au serveur et termine, sinon il se met en attente de la durée de la période.

Lorsque le serveur Tomcat est éteint, la méthode **ServerStart.contextDestroyed** est appelée et le serveur indique à TimeManager qu'il doit terminer. Ce dernier indique alors aux **GameTime** qu'ils doivent terminer leur exécution. Voir figure 6 pour le diagramme de séquence représentant ce comportement.

Cette implémentation fait que le projet est très adapté à une situation réelle ou les utilisateurs créé le contenu de la base de données, mais un peu moins au contexte ou un correcteur essaye de tester rapidement certaines fonctionnalités.
En effet, il est impossible d'ajouter des tuples dans la base de données à la main, pour les parties en cours par exemple, sous peine de corrompre l'intégrité de la base de données et avoir des parties où aucun thread n'est attribué. Il est très important de laisser le serveur lancer les parties lui même. Afin de simplifier le testing du projet, nous avons toutefois permis de créer une partie qui doit se lancer dans le passé, elle sera alors automatiquement lancée par le serveur.

4.1.3 Automatisation des requêtes : *PL/SQL*

Plusieurs fonctions, procédures et triggers ont été créé pour automatiser certaines opérations sur la base de données. Par exemple, un trigger permet de faire que lorsqu'une partie est créée, son créateur est ajouté dans gameLobby en tant que membre de la partie. Une procédure permet de faire les opérations pour attribuer les rôles aléatoirement aux utilisateurs lorsqu'une partie et d'autres procédures, fonctions et triggers sont utilisés pour gérer les votes et autres opérations.

L'utilisation du PL/SQL permet dans certains cas de simplifier le code mais également d'avoir une base de données qui se gère elle même pour rester cohérente, principalement avec l'utilisation de triggers. Les procédures et fonctions permettent principalement de garantir les propriétés ACID de la base de données.

4.2 Interprétations du sujet

Nous avons fait certain choix d'interprétations au cours du développement du projet :

1. Il est impossible d'annuler un vote proposé.
2. Le spiritualiste ne peut pas apprendre le rôle d'un joueur mort.
3. La partie peut être lancée à n'importe quel moment, y compris le passé.
4. Un joueur peut jouer à plusieurs parties en même temps, quitter la partie et y revenir à tout moment.
5. Le jeu ne révèle pas le rôle du joueur retrouvé mort (il est indiqué que "le logiciel ne doit jamais indiquer à un joueur quels pouvoirs ont les autres").
6. Un joueur spécial contaminé perd son pouvoir.
7. Un joueur participe automatiquement à une partie qu'il a créé.

De plus, nous avons interprété que les pouvoirs spéciaux de spiritualisme et de voyance ne sont disponibles que pour les humains, puisqu'il serait inutile pour un loup-garou d'avoir ce pouvoir (les loups-garous connaissent les membres de leur camp et ils ont accès à toutes les discussions). Par cohérence avec l'interprétation précédente, nous avons également considéré que la voyance était un pouvoir qui ne pouvait être attribué qu'à un être humain puique ces deux pouvoirs sont expliqués de la même manière.

4.3 Outils

Nous avons utilisé :
— `StarUML` comme modeleur UML pour réaliser les diagrammes d'analyse et de conception.
— `Trello` pour organiser notre projet et se répartir les différentes tâches.
— `Git` comme gestionnaire de version.

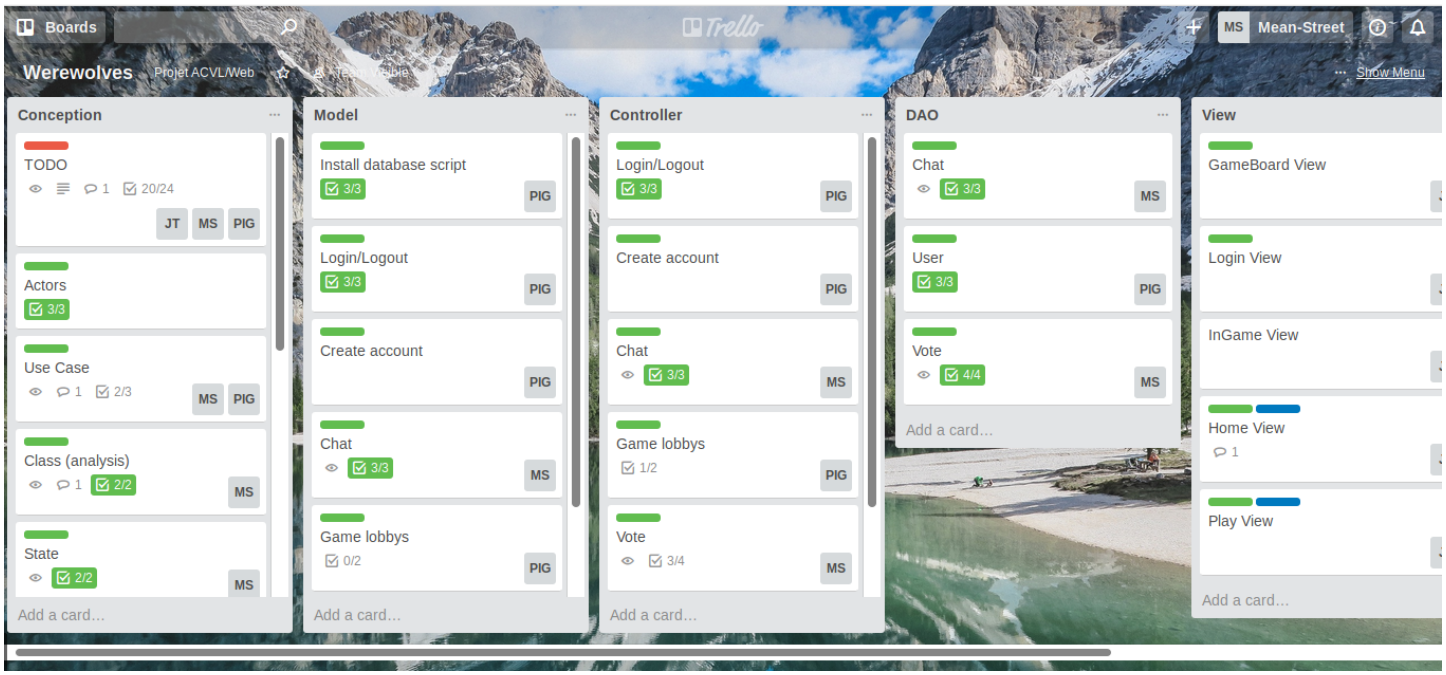

FIGURE 13 – Screenshot of our Trello organisation

4.4 Contenu du rendu

La racine du rendu contient la liste de dossier suivante :
— `conception` qui contient la liste des diagrammes réalisés au début du projet.
— `sql` qui contient :
  — `install_bd.sql` pour créer : les tables, triggers, procédures et fonctions.
  — `drop_bd.sql` pour détruire la base de données.
  — `di_bd.sql` pour réinitialiser la base de données : détruire et créer.
— `src` qui contient le projet maven contenant :
  1. Les sources côté serveur (java) : *src/main/java*
  2. Les sources côté client (html, jsp et javascript) : *src/main/webapp*

4.5 Bugs susceptibles d'être rencontrés

Le principal bug restant dans l'application est l'opération de déconnexion. Lorsque l'utilisateur clique sur le bouton de déconnexion, il est redirigé vers la page d'accueil. Après cela, le header du site devrait changer pour que le bouton gameboard disparaisse et que le bouton de connexion apparaisse. Malheureusement, le navigateur ne recharge pas la page et préfère récupérer l'ancienne version stockée dans le cache. Cela signifie qu'après chaque déconnexion, il faut appuyer sur <F5> pour actualiser la page d'accueil et obtenir la version adaptée à l'état de la session.
Une solution possible est de rediriger avec un attribut en get généré par le serveur pour forcer le navigateur à toujours redemander la dernière version et ne pas utiliser celle du cache.

Un autre bug possible est d'avoir la duplication de certaines parties ou d'avoir des boutons qui ne répondent plus dans la partie en jeu, ceci est dû à des requêtes (ajax wrappé par jquery) au serveur qui échouent. Cela est très certainement dû à un problème temporaire d'accès au serveur et il faut simplement attendre qu'il redevienne disponible.
Une dernière erreur possible est l'absence d'éléments dans la liste des parties et avoir l'impression qu'aucune

des actions ne se fait correctement. Ceci est certainement dû à une incohérence dans la base de données qui empêche le projet de se comporter correctement, avec des erreurs de tables ou autre. La meilleur solution serait alors de réinitialiser la base de données.

4.6 Instructions de déploiement

4.6.1 Serveur Apache Tomcat

La connexion à la base de données de l'Ensimag se fait avec **ojdbc7**, il faut donc inclure le fichier *src/werewolves/ojdbc7* dans le dossier *apache-tomcat/lib* pour que la connexion se fasse correctement.

Il faut également ajouter un rôle dans le fichier *apache-tomcat/conf/tomcat-users.xml* afin que Tomcat puisse se connecter à la base de données avec l'identifiant et le mot de passe souhaité :

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="carrel" password="carrel" roles="manager-gui, manager-script"/>
</tomcat-users>
```

4.6.2 Déployer le site web

Une des fonctions **PL/SQL** créées retourne un curseur et l'application doit donc pouvoir utiliser le type de données **OracleTypes.CURSOR**. Pour importer la classe **OracleTypes**, il faut inclure la dépendance **ojdbc7** dans le *pom.xml*, ce qui est déjà fait pour le projet livré.
Il est également nécessaire d'installer la librairie, et ceci doit être fait sur la machine qui va déployer l'application sur le serveur Tomcat :

```
mvn install:install-file -Dfile=ojdbc7.jar -DgroupId=com.oracle
-DartifactId=ojdbc7 -Dversion=12.1.0.1.0 -Dpackaging=jar
```

Une fois la configuration terminée, il faut lancer le serveur Tomcat avec *apache-tomcat/bin/startup.sh*, puis utiliser **Maven** pour déployer le site web sur le serveur :

`mvn tomcat:deploy` ou `mvn tomcat:redeploy`

Attention, tomcat nécessite d'avoir renseigné ses identifiants dans le fichier `~/.m2/settings.xml`. Le site web est accessible avec l'url suivante : *http ://localhost :8080/werewolves/*.