

TABLE DES MATIERES

1. PROJET.....	2
1.1. DESCRIPTION DU JEU.....	2
1.2. LES ROLES.....	2
1.2.1. LISTE NON EXHAUSTIVE DES ROLES.....	2
1.2.2. CLASSIFICATION	3
1.3. OBJECTIFS.....	4
1.4. ORGANISATION.....	4
2. ARCHITECTURE.....	5
2.1. ARCHITECTURE GENERALE.....	5
2.2. CONTENEUR DE JEU.....	5
2.3. UTILISATION DE L'AGENT DF	6
2.4. GESTION DE LA PARTIE.....	7
2.5. INITIALISATION DE LA PARTIE	8
2.6. MORT DES JOUEURS.....	9
2.7. GESTION DES ROLES.....	9
2.8. GESTION DU VOTE & IA.....	11
2.8.1. VOTE ENTRE LES JOUEURS.....	11
2.8.2. CHOIX DU JOUEUR : VOTE INTERNE.....	12
2.8.3. EXEMPLE DE SCORING.....	14
3. CONCLUSION.....	19
3.1. BILAN.....	19

1. PROJET

1.1. DESCRIPTION DU JEU

Les Loups Garous de Thiercelieux est un jeu de société dans lequel chaque joueur incarne un citoyens ou un loup-garou, et dont le but général est :

- pour les citoyens (dont certains ont des pouvoirs) : démasquer et tuer tous les loups garous ;
- pour les loups garous : d'éliminer tous les citoyens.

La partie est menée par une personne ne prenant pas part au jeu. Le meneur distribue une carte à chaque joueur, au hasard. Puis chaque joueur découvre secrètement son identité en regardant sa carte.

Les tours de jeu sont rythmés par une période de jour et une période de nuit.

Durant la nuit, tous les joueurs sont endormis. À ce moment-là, les loups garous se réveillent, appelés par le meneur de jeu, et désignent ensemble (par gestes pour éviter de se faire reconnaître) un joueur qui sera leur victime. Les villageois qui ont des capacités spéciales (cupidon, sorcière...), sont appelés pour qu'ils utilisent leurs pouvoirs respectifs.







Le jour revenu, tout le monde se réveille et ouvre les yeux. À ce moment-là, le meneur de jeu révèle l'identité de la victime des loups garous ou de la sorcière (le meneur ne doit pas le préciser). Ce joueur n'interviendra plus jusqu'à la fin du jeu et deviendra inactif. Les Villageois vont tenter de découvrir l'identité des loups garous par déductions, suspicions. Les loups garous doivent éviter de se faire accuser en détournant les soupçons sur d'autres personnes.

Il y a donc un temps de discussion au cours duquel chacun doit tenter de découvrir la véritable identité de chaque joueur.

À la fin du débat, chaque joueur vote pour le personnage qu'il suspecte. Une fois désigné par une majorité, le coupable est exécuté. À ce moment-là, le jeu recommence à la tombée de la nuit.

1.2. LES ROLES

1.2.1. LISTE NON EXHAUSTIVE DES ROLES

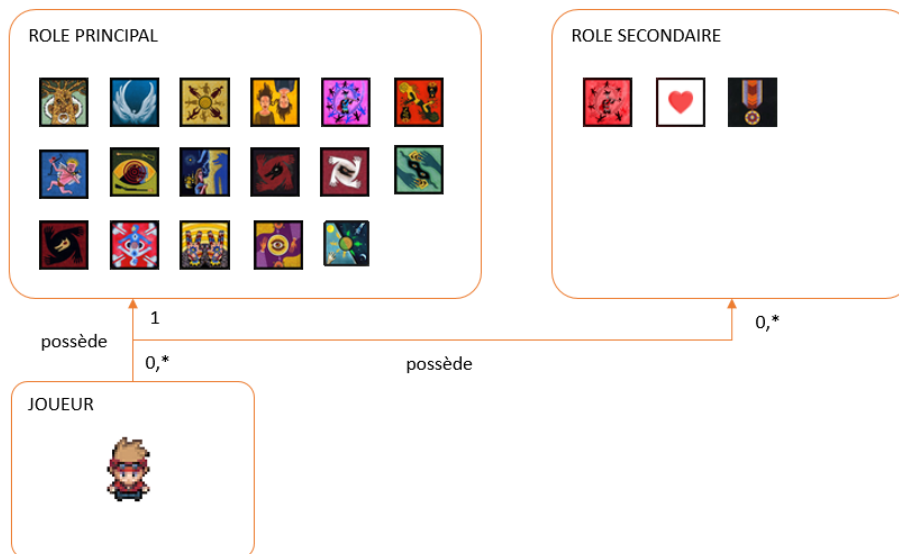
 Citizen	Une personne du village, il n'a aucun pouvoir particulier.
 Werewolf	Ennemi des villageois, il se réunit la nuit avec ses semblables et votent pour choisir une victime, qui sera tuée à la fin de la nuit.
 Great werewolf	Quand aucun loup garou n'est mort, lui et ses semblables peuvent tuer une autre victime durant la nuit.
 Loup blanc	il n'est ni avec les villageois, ni avec les loups garous. Il doit tuer tout le monde pour gagner. Il est cependant considéré comme un loup-garou par les autres loups garous ;
 Thief	Il prend le rôle d'un autre joueur au début de la partie. Le joueur volé devient un simple citoyen.
 Medium	Le médium découvre le rôle d'une personne chaque nuit.

 Little girl	Elle espionne les loups garous la nuit.
 Hunter	Lorsqu'il meurt, il peut tuer quelqu'un directement.
 Family	Les membres de la famille se réunissent durant la nuit pour se mettre d'accord sur le prochain vote d'élimination.
 Salvator	Le salvator protège chaque nuit quelqu'un.
 Cupid	Au début de la partie, il désigne deux personnes qui sont amoureuses (lover)
 Lover	Les amoureux sont liés par amour. Si l'un meurt, l'autre meurt également. Ils ont l'interdiction formelle de voter l'un contre l'autre.
 Witch	La sorcière a deux potions, l'une de vie pour soigner une victime et l'autre de mort, qui lui permet de tuer quelqu'un.
 Angel	Il doit mourir au dès le premier jour. Si c'est le cas, il gagne.
 Ancient	Il possède deux vies.
 Exorcist	Chaque nuit, il se concerte avec les morts pour voter
 Scapegoat	En cas d'égalité, le bouc émissaire prend une voix de plus s'il s'agit d'un vote « contre » et perd une voix si il s'agit d'un vote « pour »
 Flute player	Chaque nuit il choisit une nouvelle personne qu'il va charmer. Si à un moment tous les joueurs sont charmés et le joueur de flûte est en vie, il gagne.
 Charmed	Personne envouté par le joueur de flûte. Il gagne en cas de victoire du joueur de flûte.
 Mayor	Le maire a sa voix qui compte double lors des votes.

1.2.2. CLASSIFICATION

Un joueur peut posséder plusieurs rôles. Les rôles peuvent être fixés au début de la partie, ajoutés en cours, ou retirés.

Comme l'indique le schéma ci-dessous, il existe des rôles principaux et secondaires.



Il existe également une relation d'héritage entre les rôles

- Tous les rôles sont des rôles CITIZEN
- Les rôles GREAT WEREWOLF et WHITE WEREWOLF sont des rôles WEREWOLF.

1.3. OBJECTIFS

L'objectif est de proposer une simulation de ce jeu en transposant son mécanisme, sa routine dans un système multi agents.

Plusieurs modes de jeu sont envisagés :

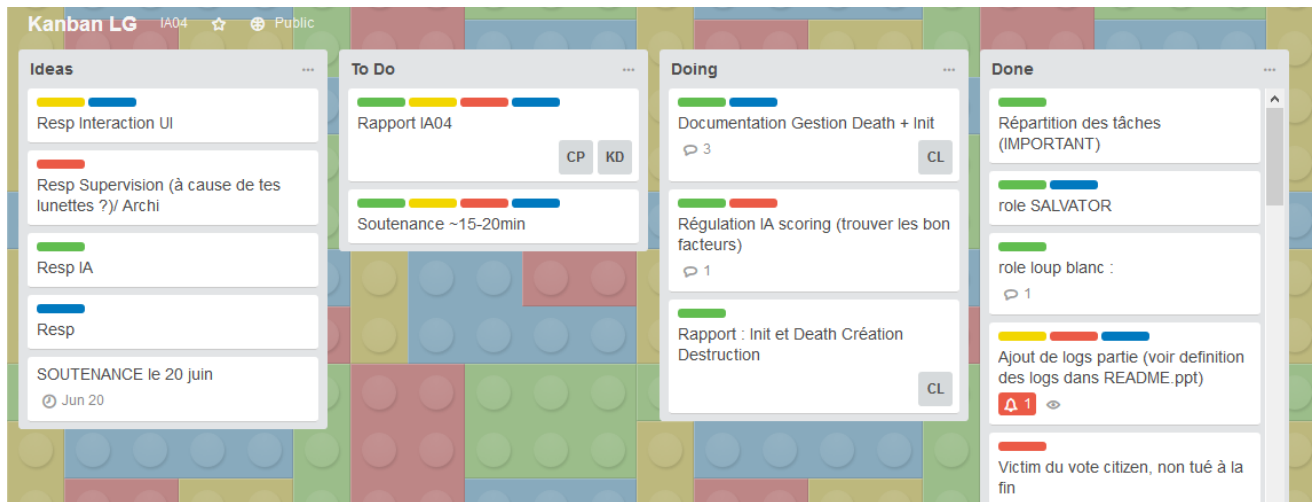
- Simulation totale : partie avec des joueurs IA. L'utilisateur ne joue qu'un rôle d'observateur.
- Simulation avec intervention humaine : un ou plusieurs joueurs sont contrôlés par l'humain.
- Mode jeu : le joueur humain ne voit pas les rôles des autres joueurs, les informations sont uniquement limitées à ce qu'il est habilité à savoir.

Le système de jeu doit pouvoir être pleinement configurable (sélection des rôles pour une partie) et il doit être possible de visualiser une partie depuis plusieurs ordinateurs.

1.4. ORGANISATION

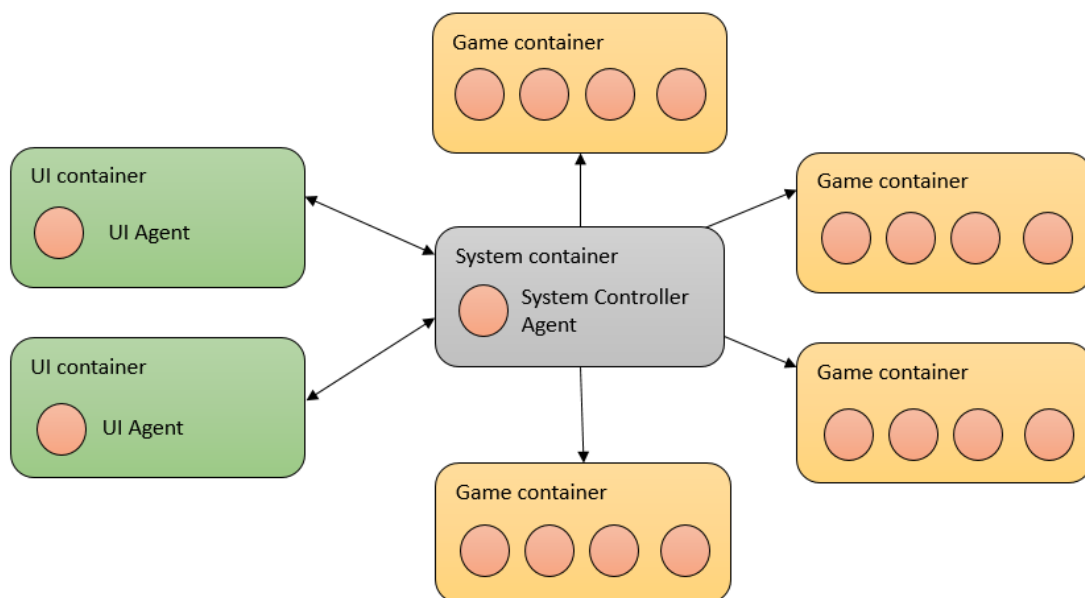
Le projet a commencé aux alentours d'Avril. L'organisation s'est faite autour du gestionnaire de version GIT et de l'outil de planification Trello.

Nous avons mis en place le principe incrémental des méthodes agiles. Le projet a été découpé en plusieurs tâches. Chaque item est soit "à faire", soit "en cours de réalisation", soit "fait". Il est traité indépendamment par l'un des membres du groupe.



2. ARCHITECTURE

2.1. ARCHITECTURE GENERALE



Notre architecture s'articule autour de trois containers :

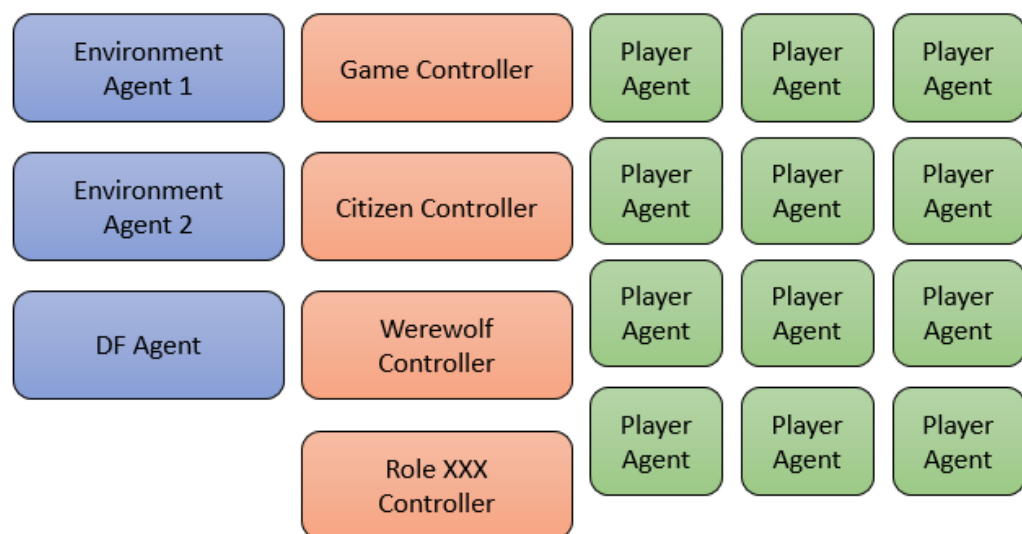
- System container : conteneur principal de l'application, il permet de lancer différentes parties de jeu via l'agent System Controller.
- Game container : conteneur de jeu. Il contient les agents nécessaires au bon déroulement du jeu. Il est créé par le System container. Il y'a autant de Game container que de partie en cours.
- UI container : conteneur spécifique à l'ihm. Il y'a autant de UI container que d'IHM utilisateur.

2.2. CONTENEUR DE JEU

Afin de distinguer les parties entre elles, les conteneurs de jeu possèdent un identifiant unique donné lors de la création du conteneur.

Le conteneur de jeu possède plusieurs agents :

- 2 agents d'environnement qui stockent les informations de la partie (résultat de vote, nombre de tours, etc...). Ils sont au nombre de deux afin d'ajouter de la redondance au système. (Il y'a une synchronisation des données pour garantir la cohérence des informations)
- Un contrôleur de jeu qui coordonne (de manière abstraite) la partie.
- Des agents contrôleurs de rôles qui gèrent un tour spécifique à un rôle.
- Des agents joueurs. Ce sont des agents génériques qui ont des comportements différents suivant le/les rôles qu'ils possèdent.



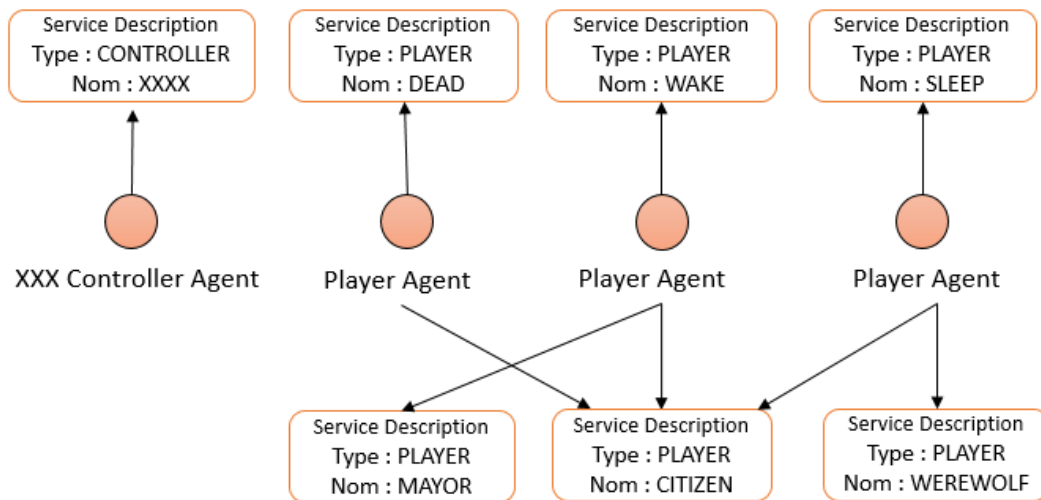
Ainsi, la taille de notre SMA est variable. Elle dépend du nombre de joueurs, mais aussi du nombre de rôles qui a été configuré.

2.3. UTILISATION DE L'AGENT DF

Les différents agents s'enregistrent auprès de l'agent DF. L'identifiant de la partie est ajouté dans la description de service pour rendre les conteneurs de jeu totalement opaque et ainsi éviter de confondre des agents de parties différentes.

Nous nous servons également de l'enregistrement via le DF pour « stocker » certaines données de jeu. En effet les joueurs s'enregistrent suivant :

- Leurs différents rôles.
- Leurs statuts (WAKE, SLEEP, DEAD)

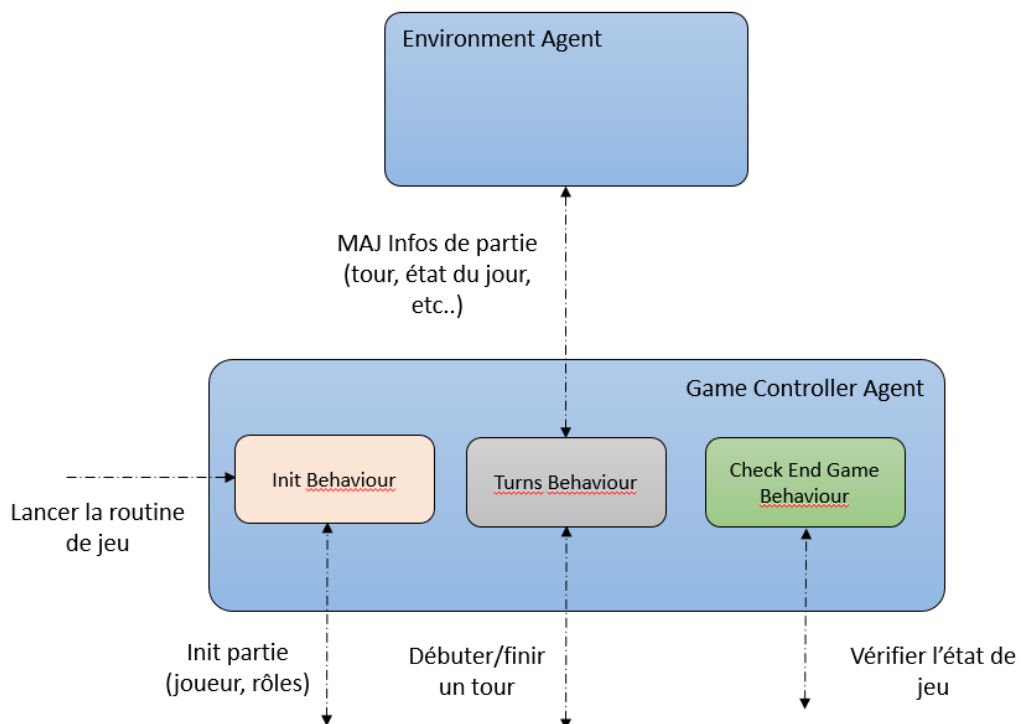


Grâce à cette façon de faire et à la surcouche développée pour l'enregistrement, il est très simple de rechercher les joueurs.

Dans l'exemple ci-dessous : nous faisons une recherche pour récupérer les loups garous reveillés.

```
String [] args = {Roles.WEREWOLF, Status.WAKE};
List<AID> werewolves = DFServices.findGamePlayerAgent(args, this.ctrlAgent,
this.ctrlAgent.getGameid());
```

2.4. GESTION DE LA PARTIE

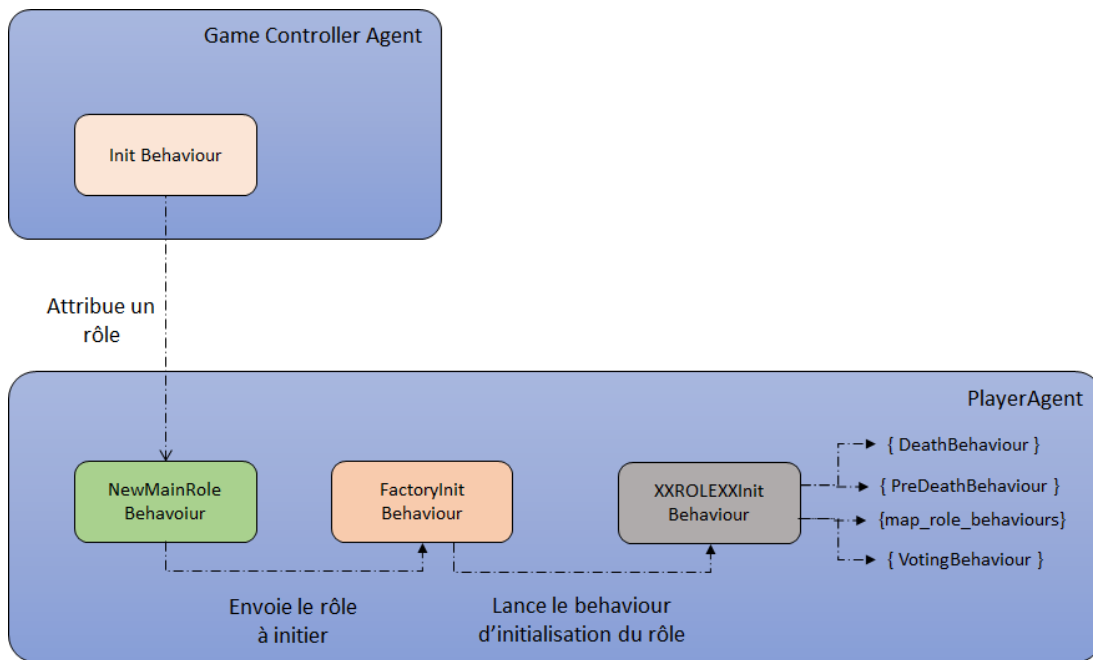


L'agent de gestion de jeu est le GameControllerAgent, il est composé de plusieurs behaviours :

- Init Behaviour, qui permet le démarrage du jeu (configuration, initialisation des joueurs)
- TurnsBehaviour qui assure la routine de jeu (déroulement des tours)
- CheckEndGameBehaviour permettant de tester la fin de jeu.

C'est un agent qui communique beaucoup avec l'agent d'environnement afin de lui fournir des infos relatives à la partie (nom du tour, nombre de tours, état du jeu).

2.5. INITIALISATION DE LA PARTIE

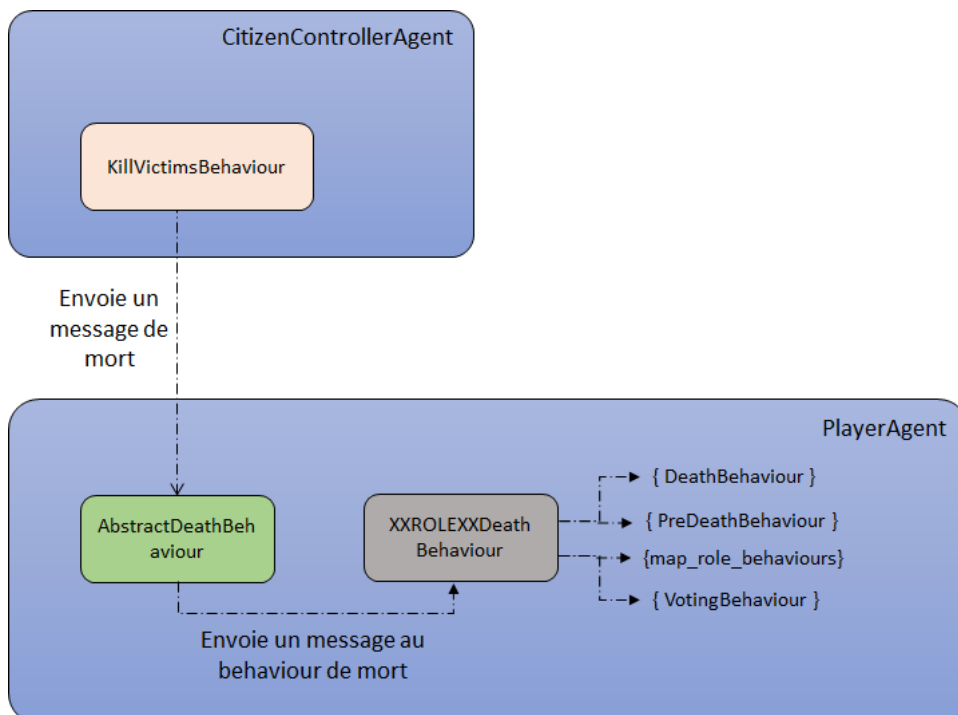


L'initialisation des rôles de chaque joueur se fait à l'aide du comportement **NewMainRoleBehaviour** du **PlayerAgent**, qui recevra le nouveau rôle à ajouter au joueur. Le rôle principal du joueur sera mis à jour s'il n'en possédait pas avant.

Ensuite un message sera envoyé au **factoryInitBehaviour**, un comportement générique qui selon le rôle reçu créera le comportement d'initialisation du rôle et le lancera dans le joueur.

L'in

2.6. MORT DES JOUEURS

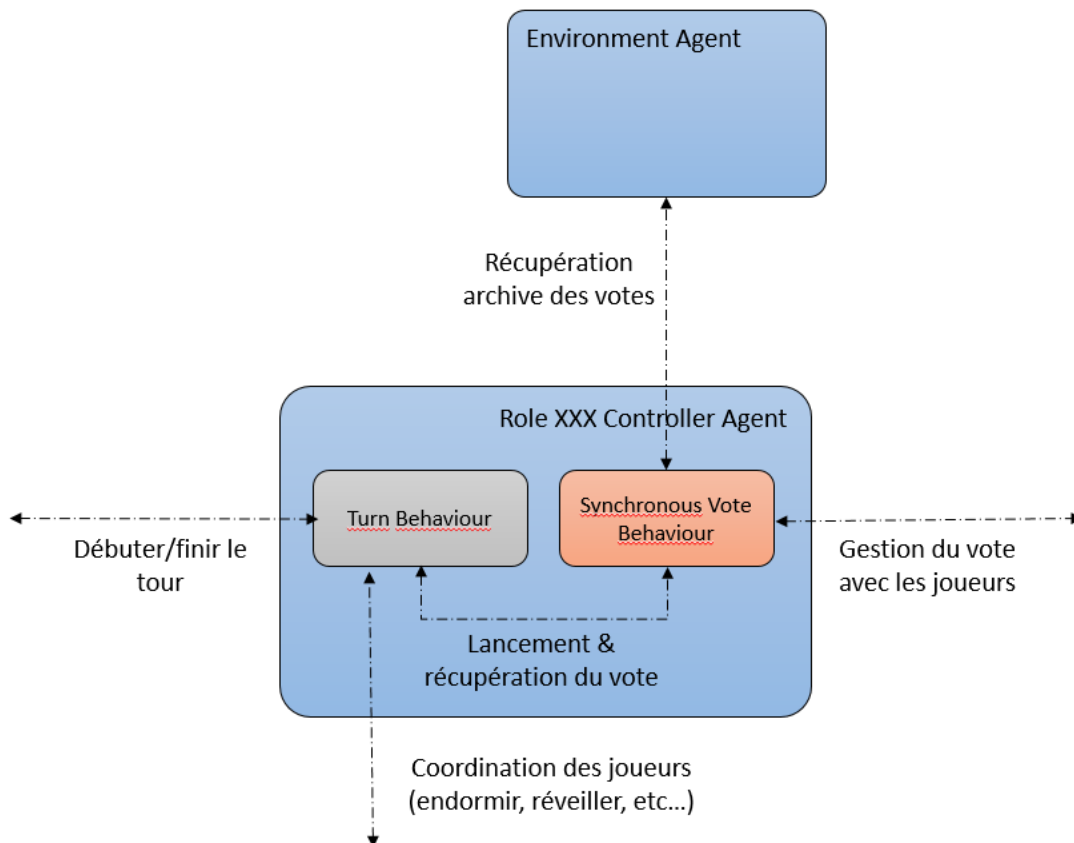


2.7. GESTION DES ROLES

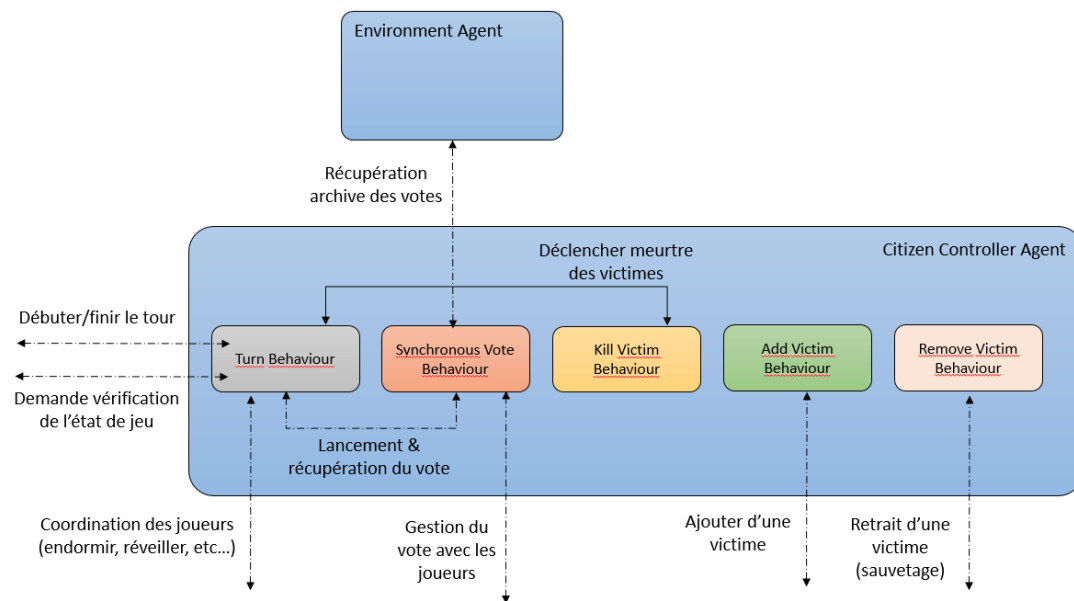
La gestion de rôles et le déroulement concret des tours sont sous-traités par le GameControllerAgent, il s'agit de la responsabilité des contrôleurs de rôles.

Ceux-ci ont une structure similaire :

- TurnBehaviour, qui permet le déroulement concret du tour
- SynchronousBehaviour, qui coordonne le jeu pendant les phases de vote.



Le plus important des contrôleurs de rôles est le CitizenControllerAgent.



2.8. GESTION DU VOTE & IA

2.8.1. VOTE ENTRE LES JOUEURS

La gestion du vote, phase cruciale du jeu, se déroule de manière synchrone, c'est-à-dire chacun des joueurs vote successivement (ce qui leurs permet de voter en fonction du résultat partiel).

Cependant, ce vote synchrone peut prendre énormément de temps lorsque le nombre de votant est important, nous avons donc fait une optimisation en permettant **le vote par « paquet »**, c'est-à-dire qu'au mieux de faire voter les joueurs un par un, nous les faisons voter petit groupe par petit groupe).

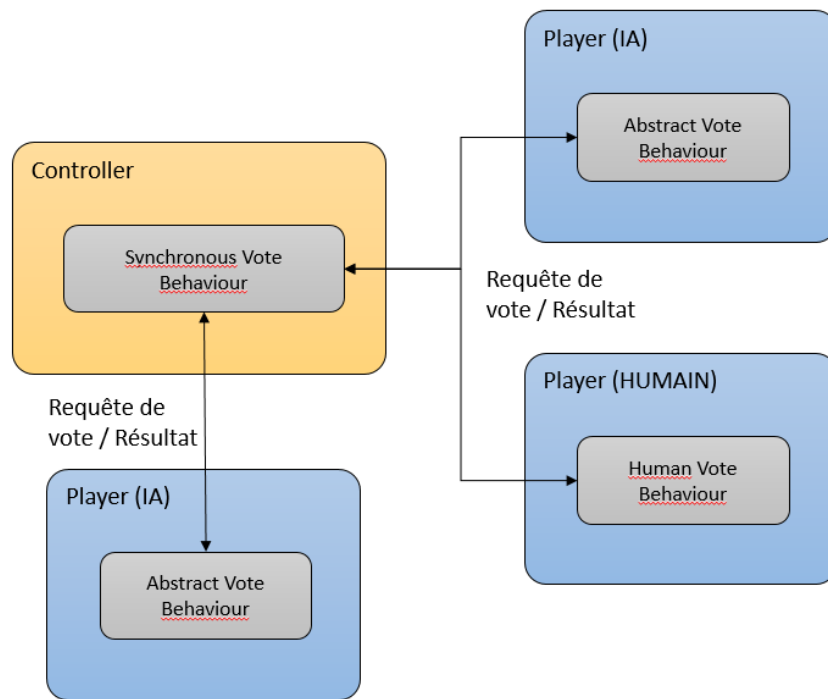
Plusieurs types de votes sont possibles :

- Vote « pour » : le joueur doit favoriser quelqu'un
- Vote « contre » le joueur doit défavoriser un autre joueur (un type de vote qui conduit très souvent à l'élimination du joueur désigné).
- Demander si le joueur souhaite voter : utile lorsqu'il y'a des actions qui se produisent seulement lorsqu'un joueur est sûr de son choix (exemple : rôle WITCH).

La désignation se fait suivant le processus suivant :

- Le joueur ayant recueilli le plus de voix est désigné.
- Si il y'a une égalité, un nouveau vote est organisé en réduisant la liste des choix possibles.
- Si il y'a une situation d'interblocage (chacun des joueurs restent sur leurs choix initiaux, provoquant de nouveau une égalité), le joueur désigné est choisi aléatoirement.

Comme le montre le schéma suivant, la gestion de vote est gérée par **SynchronousVoteBehaviour** : behaviour présent dans chacun des agents contrôleurs de rôles.



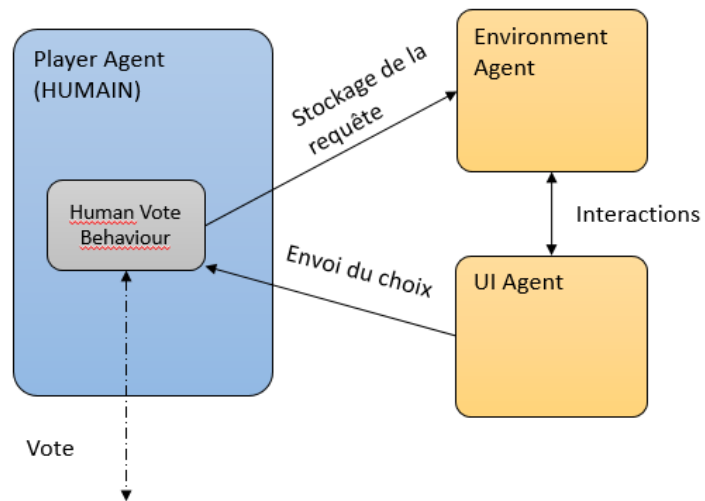
2.8.2. CHOIX DU JOUEUR : VOTE INTERNE

Lors d'un vote, les choix des joueurs sont complètement différents :

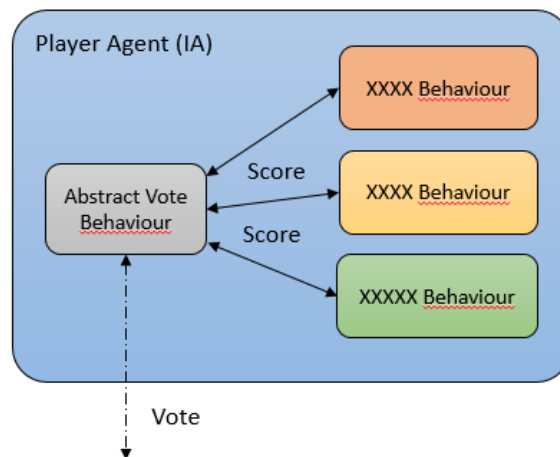
- Quand le joueur est humain, c'est à l'utilisateur d'indiquer pour qui il souhaite voter.
- Quand le joueur est une IA, il doit avoir un choix cohérent avec les rôles qu'il possède.

Ainsi, le comportement de vote est différent entre les joueurs humains et IA.

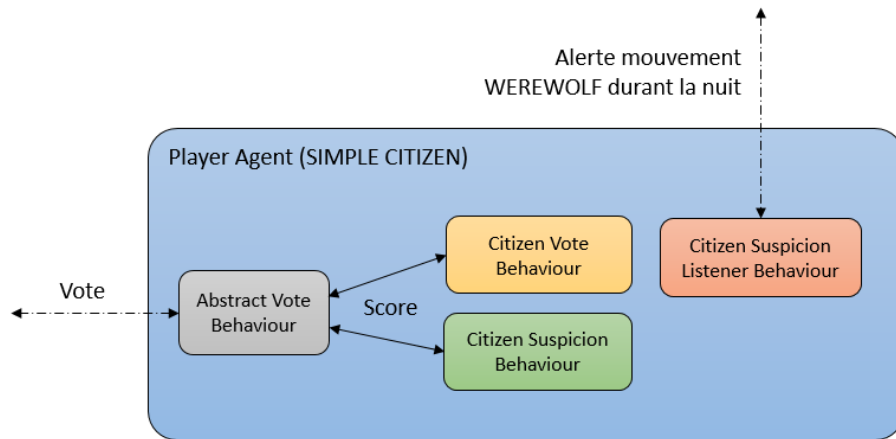
Les joueurs humains stockent la requête de vote dans l'agent d'environnement, lorsqu'ils en reçoivent une ; afin que celle-ci s'affiche dans l'IHM et que l'utilisateur puisse voter.



Pour proposer un choix pertinent en fonction des rôles, la méthode scoring a été retenue. L'intelligence de chacun des rôles va être encapsulée dans une ou plusieurs unités de réflexion.



Ci-dessous un exemple représentation interne en termes d'unités de réflexions.



2.8.3. EXEMPLE DE SCORING

Les 2 exemples de scoring que nous allons étudier sont issues d'un déroulement du jeu. Ils représentent les scores associés au vote du joueur E, lorsqu'il souhaite éliminer une personne.

Ce joueur possède plusieurs rôles :

- **FLUTE_PLAYER** : son but est de charmer l'ensemble des joueurs vivants. Il remporte le jeu s'il réussit cet objectif. A ce stade de la partie, les joueurs A et D ont déjà été charmés.
- **LOVER** : au cours de la partie, le joueur CUPIDON la désigné pour être amoureux avec le joueur A. Si l'un meurt, l'autre aussi.

Afin de mettre en valeur le raisonnement de l'IA, nous allons montrer les scores générés lorsque le joueur possède davantage de point de capacités pour sa réflexion stratégique, puis lorsqu'il y'a davantage de points sur la réflexion suspicieuse.

Type stratégique	3 / 5
Type suspicieux	2 / 5

	Joueur A	Joueur B	Joueur C	Joueur D	Joueur E (Votant)
IA Role CITIZEN	1	2	0	1	-1000
IA Role LOVER	-500	0	0	0	0
IA Role FLUTE_PLAYER	-10	4	1	-2	0
IA Suspicion	0	0	5	1	0
	-1527	18	13	-1	-3000

Dans ce premier exemple, il s'agit du joueur B qui a été désigné (score maximum).

- Afin que le joueur E ne vote pas pour lui-même (s'agissant d'un vote d'élimination, ça serait du « suicide »), l'unité de réflexion du rôle CITIZEN lui a affecté un grand score négatif.
- Pour éviter de voter pour son amoureux, l'unité de réflexion associé au rôle LOVER a également effectué un score négatif au joueur A.

Type stratégique	2 / 5
Type suspicieux	3 / 5

	Joueur A	Joueur B	Joueur C	Joueur D	Joueur E (Votant)
-IA Role CITIZEN	1	2	0	1	-1000
IA Role LOVER	-500	0	0	0	0
IA Role FLUTE_PLAYER	-10	4	1	-2	0
IA Suspicion	0	0	5	1	0
	-1018	12	17	1	-2000

Ici la réflexion stratégique possède un poids plus important et cela fait la différence par rapport à la configuration précédente.

Le joueur vote ici contre le joueur C, contre qui il a le plus de soupçon.

Nous constatons que la répartition des points de capacités influence fortement le vote du joueur, cela permet ici d'avoir des joueurs aux raisonnements différents même si ils possèdent exactement les mêmes rôles.

2.7 IHM

2.7.1 ARCHITECTURE GLOBALE

Le but de l'IHM ici est double :

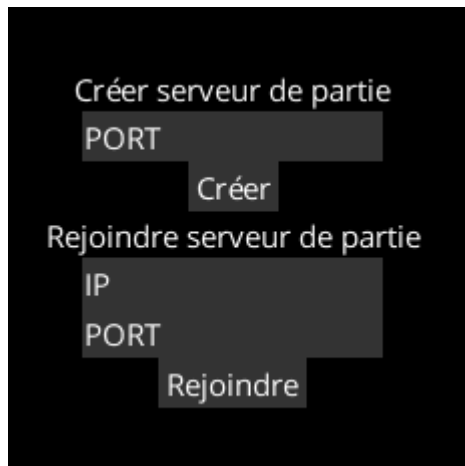
- Faciliter l'accès des utilisateurs aux multiples parties proposées sur un serveur choisi.
- Permettre une visualisation des échanges de messages et leurs conséquences car la visibilité proposée par Jade ne suffit pas.

Pour cela, l'IHM est donc décomposé en 2 grandes parties, la première comprend un ensemble d'écrans permettant l'accès à la création ou à la visualisation d'une partie. La seconde est l'écran de jeu.

2.7.2 INTERFACE D'ACCES

2.7.2.1 L'ECRAN PRINCIPAL

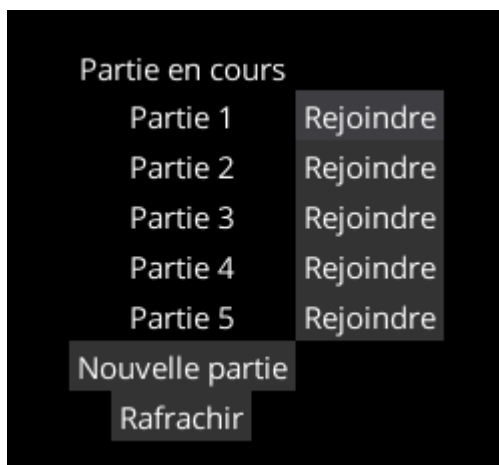
Lors du lancement de l'application, l'utilisateur a la possibilité de créer un serveur ou d'en rejoindre un. Dans les deux cas, il devra indiquer le port utilisé souhaité.



Dans les deux cas, un UI container va être rattaché au conteneur principal, le System container, disponible à l'IP et port indiqués. La différence lorsque l'on appuie sur le bouton de création de serveur par rapport au bouton rejoindre, c'est qu'au préalable, un System container va être créé et que l'UI container va utiliser automatiquement l'IP locale pour la liaison.

2.7.2.2 VISIBILITE DES PARTIES

Une fois le serveur joint, une liste des parties actives sont affichés avec un bouton pour les rejoindre, il y a également un bouton permettant d'accéder à la fenêtre de création de parties.



Les boutons sont directement liés à l'id de la partie pour faciliter les échanges de messages.

2.7.2.3 CREATION DE PARTIES

La création de partie se fait via un écran permettant d'indiquer le nombre de joueurs pour chaque rôle, le nombre de joueur humain ou de cacher les rôles. Toutes ces informations sont stockées dans un objet qui est utilisés à l'initialisation d'un nouveau Game Container.

Werewolf			
Normal	0	Great	0
White	0		
Citizen		Cupid	0
Normal	0	Medium	0
Family	0	Hunter	0
Little girl	0	Flutist	0
Angel	0	Witch	0
Ancient	0		
Thief	0		
Joueurs humains			
Nombre :	0		
<input type="checkbox"/> Cacher les roles			
<div>Créer partie</div> <div>Retour</div>			

2.7.2.4 AFFICHAGE PARTIE

Cette partie contient toutes les informations nécessaires pour suivre le déroulement de la partie.

1 Jour

2 Citizen

3 PLAYER_0_B@main_container

Vote

4 Retour

7

5

PLAYER_0_V se reveille
 PLAYER_0_Q se reveille
 PLAYER_0_I se reveille
 PLAYER_0_B se reveille
 Mort des victimes
 PLAYER_0_S r2flgchit
 PLAYER_0_S vote PLAYER_0_S
 PLAYER_0_N r2flgchit
 PLAYER_0_N vote PLAYER_0_E

6 Vote

PLAYER_0_S : 1
 PLAYER_0_E : 1
 PLAYER_0_D : 0
 PLAYER_0_I : 0
 PLAYER_0_B : 0
 PLAYER_0_U : 0
 PLAYER_0_A : 0
 PLAYER_0_W : 0

Indique le temps : jour ou nuit.

Indique le tour des joueurs pour un rôle spécifique.

Utiliser dans le cas où un joueur est contrôlé par un humain, le bouton Vote envoie la réponse sélectionnée.

Permet de retourner à l'écran de sélection de la partie.

5)

3. CONCLUSION

3.1. AMELIORATIONS

Plusieurs pistes d'optimisations et d'améliorations sont possibles :

- Prise en compte des différentes extensions du jeu afin de proposer de nouvelles règles (l'extension Village permet par exemple l'ajout de métiers)
- Fonctionnalité de sauvegarde
- Optimisation du système multi agents via l'application d'algorithmes répartis vu en SR05

3.2. BILAN

Ce travail commencé en Avril et poursuivi tout au long du semestre s'est achevé sur une application fonctionnelle.

Nous avons rencontrés de nombreux problèmes, notamment dans le débogage de nos behaviours qui s'est avéré difficile en raison de la complexité du système.

Néanmoins il s'agit d'une architecture solide, fiable et évolutive. Il est très facile d'ajouter de nouveau rôle.

Enfin, ce projet nous a permis d'approfondir les notions vues en cours d'IA04.