Master 1 GIL - Gestion de projet
Document d'Architecture Logiciel
Agora V3-1

Groupe 1 Agora V3-1 : Partie bibliothèque de jeux

13 mars 2024

Version	8
Date	13 mars 2024
Rédigé par	BINGINOT Etienne CAUCHOIS Niels DUCROQ Yohann KHABOURI Izana MAZUY Axelle MONTAGNE Erwann THIBERVILLE Malvina VAN LIEDEKERKE Florian

Mises à jour du document

Version	Date	Modification réalisée
1	29 novembre 2023	Création du document
2	11 décembre 2023	Ajout du GameService
3	17 décembre 2023	Ajout des logs
4	27 décembre 2023	Mise à jour des DTO
5	22 janvier 2024	Mise à jour après les retours de l'enseignant de gestion de projet
6	25 février 2024	Suppression du GameController non factorisable Renommage du contrôleur ChatController en MessageController
7	12 mars 2024	Ajout de la partie architecture front-end
8	13 mars 2024	Mise à jour du GameService en AbstractGameManagerService

Table des matières

1 Objet du document

Ce document a pour but de décrire l'architecture logicielle du projet Agora. Nous allons décrire l'architecture de la bibliothèque de code.

Notre équipe s'occupant de refaire la bibliothèque de jeux, plusieurs contraintes nous sont imposées :

— créer une bibliothèque commune de code utile à la création de n'importe quel jeu, cela permettra de factoriser le code et de pouvoir créer un nouveau jeu assez facilement ;

— rédiger une documentation claire et complète, ce qui permettra aux futurs développeurs de pouvoir intégrer de nouveaux jeux facilement, en expliquant en détails notre bibliothèque de code et comment l'utiliser.

L'enjeu majeur est donc de bien analyser et trouver ce qu'il y a en commun entre tous les jeux.

D'un point de vue technique, l'utilisation du framework Symfony, et donc du langage PHP, est primordiale.

Il est aussi attendu de pouvoir intégrer parfaitement notre système de jeux dans la plateforme, côté site, réalisée par le second groupe sur AGORA-Lab.

Enfin, le but final de notre projet est de disposer d'une bibliothèque de code factorisée ainsi que d'une documentation claire et complète. Ces deux éléments nous permettant de créer 4 jeux (6 qui prend, Splendor, Myrmes, Glenmore), demandés par le client, qui devront être jouables sans aucun problème.

2 Documents de références

Nos références utilisées sont les DAL des précédentes années concernant le projet AGORA, la feuille de présentation du projet, les STB réalisées ainsi que les compte-rendus des réunions avec le client et le référent technique.

3 Terminologie

Joueur : rôle que peut prendre un utilisateur humain lorsqu'il participe à un jeu, lui permettant de discuter avec les autres joueurs dans le chat et de jouer son tour.

Spectateur : rôle que peut prendre un utilisateur humain lorsqu'il n'est pas joueur. Permet d'observer l'ensemble des plateaux d'une partie.

Partie : jeu en cours, une fois les paramètres et règles déterminés, conférant une action au tour par tour à chaque joueur.

framework : les frameworks sont conçus et utilisés pour modeler l'architecture des logiciels applicatifs, applications web et des composants logiciels.

Symfony : c'est un ensemble de composants PHP qui fournit des fonctionnalités modulables et adaptables ce qui permet de faciliter et d'accélérer le développement d'un site web.

PHP : c'est un langage de programmation libre principalement utilisé pour produire des pages Web dynamiques.

4 Configuration requise

4.1 Matériel demandé aux joueurs pour utiliser l'application

Aucune configuration n'est réellement requise concernant l'utilisation de l'application Agora, le joueur devant pouvoir jouer sur ordinateur, téléphone ou tablette.

4.2 Configuration en environnement de déploiement

Afin de déployer le projet Agora, les technologies suivantes sont nécessaires :
— Ubuntu en version 22.04 (LTS : Long Time Support) (STB-REA-11)
— PHP en version 8.3.0, avec le plugin opcache (STB-REA-02)
— Tailwind installé en version 3.4.1 (STB-REA-10)
— MariaDB en version 10.11.2 (STB-REA-05)
— Symfony installé en version 6.4.2 (STB-REA-01)
— Apache en version 2.4.58 (STB-REA-03)

4.3 Configuration en environnement de développement

Pour le développement de l'application Agora, nous utiliserons :

— Les mêmes technologies que les configurations en environnement de déploiement 4.2
— PHPUnit en version 9.6.13 (STB-REA-04)
— GitLab en version 16.5.1 (STB-REA-06)
— Docker en version 24.0.7 (STB-REA-07)
— SonarQube en version 9.9 (STB-REA-08)
— SonarLint version dépendante de l'IDE (STB-REA-09)

5 Architecture statique

Dans l'architecture statique de ce document, nous retrouvons particulièrement les composants qui ont été identifiés comme récurrents dans l'ensemble des documents spécifiques aux jeux concernant l'architecture.

5.1 Structure générale

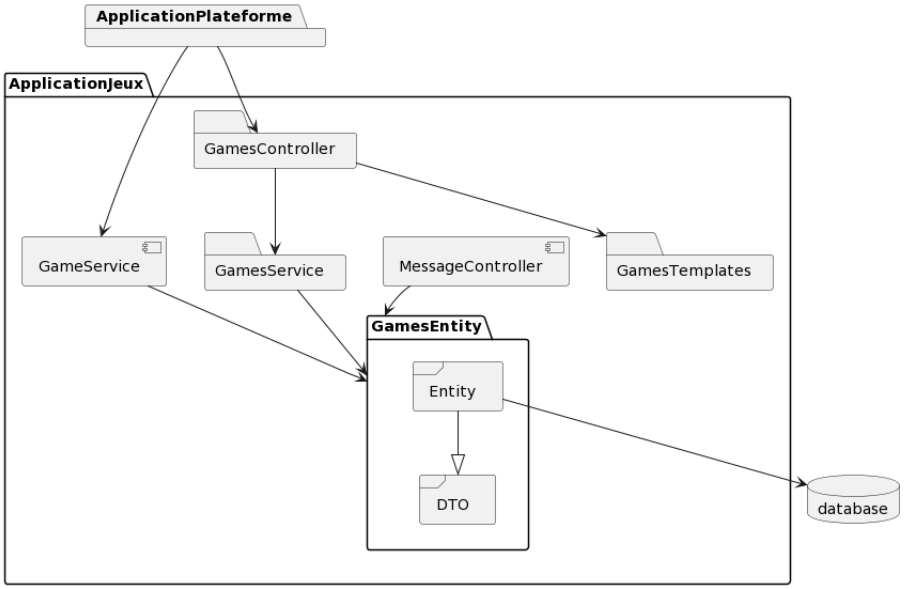

FIGURE 1 – Schéma d'architecture général

Sur ce schéma nous pouvons voir figurer différents éléments :

— ApplicationJeux : L'application jeux est l'objet représentant le code de l'application web du point de vue des jeux, autrement dit le code des différents jeux dans notre cas. Le code est écrit en PHP avec le framework Symfony. On peut retrouver dans cet élément :

- Un composant GameService : C'est ce composant qui va se charger de créer une partie de jeu avec une liste de joueurs. Il pourra également ajouter ou enlever des joueurs d'une partie, arrêter (en conservant les logs) cette dernière ou la démarrer.
- Un composant MessageController : C'est ce composant qui va se charger de gérer les messages reçus et envoyés par les joueurs dans le chat de discussion.
- Un dossier GamesController : Ce dossier regroupe l'ensemble des contrôleurs qui vont permettre de gérer les actions que les joueurs peuvent réaliser au cours de la partie, pour chaque jeu. Les contrôleurs permettent de définir les routes de notre application web.
- Un dossier GamesService : Ce dossier regroupe l'ensemble du code métier du projet qui va manipuler les entités, pour chaque jeu.
- Un dossier GamesEntity : Ce dossier regroupe l'ensemble des objets représentant un jeu, qu'ils soient des DTO ou en base de données.
- Un dossier GamesTemplates : Ce dossier regroupe l'ensemble des fichiers de code représentant le visuel de l'application, pour chaque jeu. Les fichiers de ce dossier sont écrits avec Twig (moteur de templates de Symfony), en utilisant également Tailwind pour le CSS.

— database : Il s'agit de la base de données de l'application web, qui utilise MariaDB. L'entièreté des entités et des logs (STB-OPE-01) sont donc stockés dedans. Cette base de données est commune à la plateforme et aux jeux, bien que les tables soient bien distinctes.

5.1.1 Mercure

Mercure est un protocole permettant d'envoyer des données aux navigateurs sans avoir besoin de rafraîchir la page de façon efficace.

Nous pouvons donc utiliser ce principe pour publier à tous les joueurs les actions réalisées par les autres joueurs en temps quasiment réel.

Ainsi l'application web se met à jour automatiquement sans intervention du client.

5.1.2 GamesTemplates

Les fichiers de vue de l'application concernant les jeux sont présentes dans le dossier templates/Game. Chaque jeu possède son dossier propre, tel que Glenmore, Splendor, Six_qp et Myrmes. Ces dossiers sont à leur tour subdivisés en catégories spécifiques.

Chaque fichier possède l'extension .html.twig. Chaque dossier de jeu comporte également un fichier index qui étend un certain fichier base.

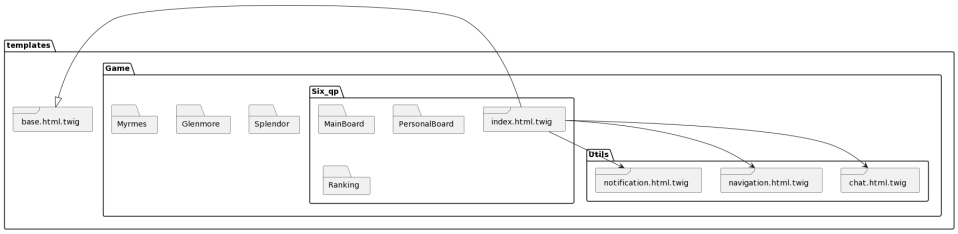

FIGURE 2 – Diagramme d'architecutre front-end

On peut voir sur le diagramme ci-dessus qu'un jeu contient 3 dossiers : MainBoard, contenant les fichiers utiles pour l'affichage du plateau principal, PersonalBoard contenant les fichiers utiles pour l'affichage du plateau personnel, Ranking contenant les fichiers du classement. On a également un autre dossier dans Game du nom de Utils, qui contient les fichiers utiles pour l'ensemble des jeux comme la barre de navigation, le chat de discussion des joueurs, et le fichier de notifications.

Chaque jeu respecte donc la même architecture.

5.2 Description des composants

5.2.1 MessageController

Ce composant permet de réunir l'ensemble des routes accessibles par les utilisateurs concernant le chat de discussion entre les joueurs lors des jeux.

Méthode	Type de retour	Description
sendMessage()	ResponseHTML	Cette route renvoie le code HTML de la partie identifiée par l'entier gameId
receiveMessage()	ResponseHTML	Cette méthode permet de recevoir les messages publiés par les joueurs et renvoyer le code HTML affichant l'historique mis à jour des messages.

5.2.2 AbstractGameManagerService

Ce composant permet de rassembler toutes les méthodes gérant une partie de jeu peu importe le jeu. Il s'agit d'une classe abstraite, implémentée par XGameManagerService, où X désigne SixQP, Splendor, Glenmore ou Myrmes.

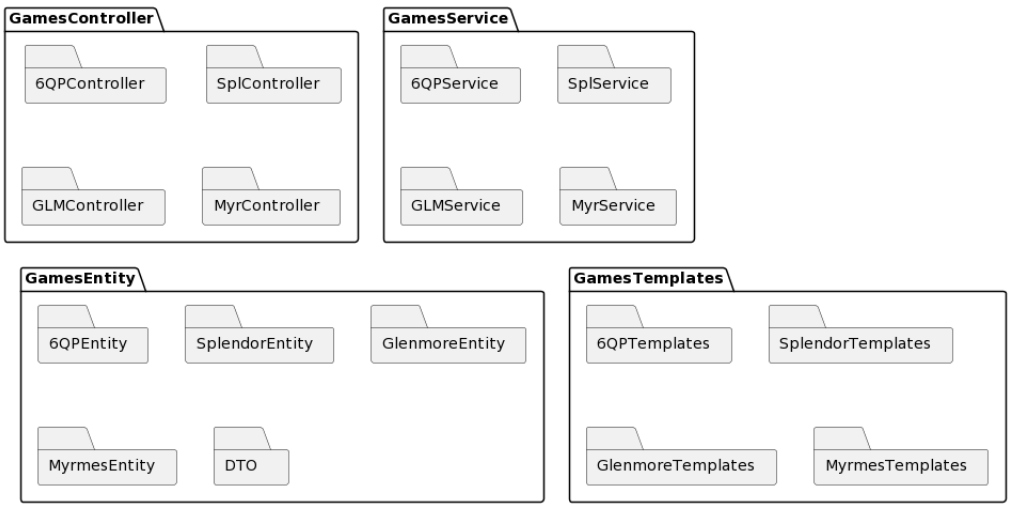

FIGURE 3 – Schéma d'architecture du détail des dossiers

Il contient pour l'instant les méthodes suivantes :

Méthode	Type de retour	Description
createGame()	int	Cette méthode permet de créer une partie du jeu sur lequel la méthode est appelée. Renvoie l'ID de la partie créée
joinGame(string playerName , Game gameId)	int	Cette méthode permet de faire rejoindre l'utilisateur de nom playerName qui est son compte utilisateur(STB-OPE-02). dans la partie identifiée par game. Renvoie un code de retour.
deletePlayer(string playerName , User user)	int	Cette méthode permet d'enlever user de la partie identifiée par gameId s'il s'agit d'un joueur de la partie. Renvoie un code de retour.
deleteGame(Game game)	int	Cette méthode permet de supprimer la partie identifiée par l'entier gameId. Renvoie un code de retour.
launchGame(Game game)	int	Cette méthode permet de lancer la partie identifiée par l'entier gameId si elle respecte les préconditions de lancement (nombre de joueurs dans la partie pour le jeu correct). Renvoie un code de retour.

Les méthodes partagent la même sémantique des codes de retour (sauf pour createGame) :
— SUCCESS : 1 : La méthode s'est déroulée correctement
— ERROR_ALREADY_IN_PARTY : -1 : L'utilisateur est déjà dans la partie
— ERROR_INVALID_GAME : -2 : La partie n'existe pas
— ERROR_GAME_ALREADY_LAUNCHED : -3 : La partie est déjà lancée
— ERROR_INVALID_NUMBER_OF_PLAYER : - 4 : Le nombre de joueurs est invalide
— ERROR_PLAYER_NOT_FOUND : -5 : L'utilisateur n'est pas reconnu

5.2.3 GameManagerService

Ce composant permet de rassembler toutes les méthodes gérant une partie.
Il s'agit d'une classe appelant AbstractGameManagerService selon les paramètres qui lui sont donnés.

Il contient pour l'instant les méthodes suivantes :

Méthode	Type de retour	Description
createGame(string gameName)	int	Cette méthode permet de créer une partie du jeu identifiée par le nom gameName. Renvoie l'ID de la partie créée
joinGame(int gameId, User user)	int	Cette méthode permet de faire rejoindre l'utilisateur user dans la partie identifiée par gameId. Le joueur créé aura le même pseudo que son compte utilisateur(STB-OPE-02). Renvoie un code de retour.
quitGame(int gameId, User user)	int	Cette méthode permet d'enlever user de la partie identifiée par gameId s'il s'agit d'un joueur de la partie. Renvoie un code de retour.
deleteGame(int gameId)	int	Cette méthode permet de supprimer la partie identifiée par l'entier gameId. Renvoie un code de retour.
launchGame(int gameId)	int	Cette méthode permet de lancer la partie identifiée par l'entier gameId si elle respecte les préconditions de lancement (nombre de joueurs dans la partie pour le jeu correct). Renvoie un code de retour.

5.2.4 Help

Nom de la classe : Help (STB-INT-05)

Attributs	Méthodes
gameName : une chaîne de caractères indiquant le nom du jeu.	
title : une chaîne de caractères indiquant le titre de la fenêtre d'aide.	
description : une chaîne de caractères contenant l'aide explicative au joueur.	
image : une chaîne de caractères correspondant à l'image associée à l'aide.	
componentId : un entier référençant le composant auquel l'aide est associée.	

5.2.5 Message

Nom de la classe : Message (STB-GES-01, STB-GES-02, STB-GES-03)

Attributs	Méthodes
id : un entier identifiant le message.	
content : une chaîne de caractères contenant le message.	
authorId : un entier identifiant le joueur à l'origine du message.	
date : une date correspondant à l'heure d'envoi du message.	

5.2.6 Log

Nom de la classe : Log (STB-OPE-01)

Attributs	Méthodes
id : un entier identifiant le log.	
gameId : un entier identifiant la partie de jeu associée.	
playerId : un entier identifiant le joueur à l'origine du log.	
message : une chaine de caractères décrivant le log.	
date : une date représentant la date d'action du log.	

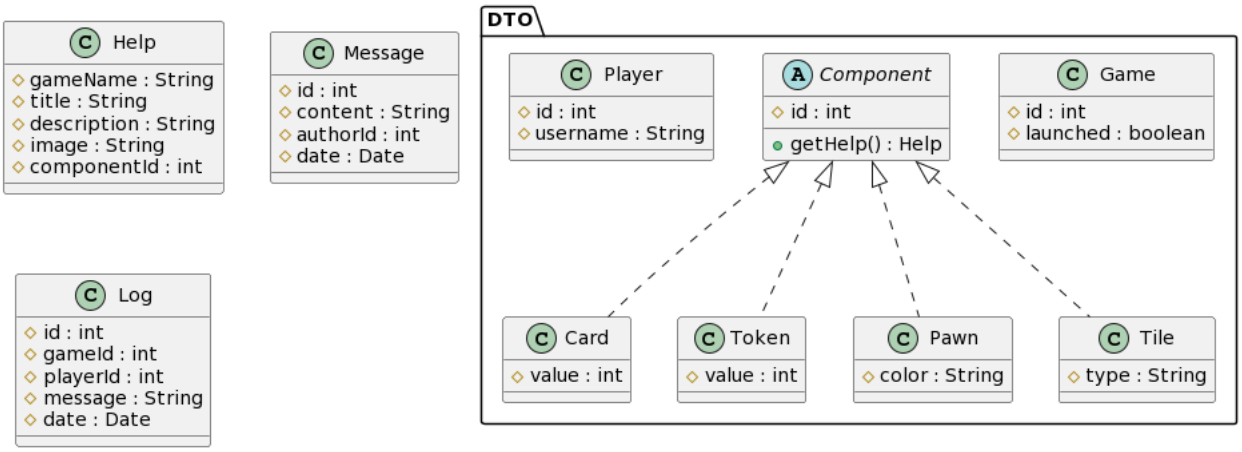

FIGURE 4 – Diagramme de classe des DTO

5.2.7 DTO

Nom de la classe : Player

Attributs	Méthodes
username : une chaîne de caractère identifiant un joueur	

Nom de la classe : Component (Abstract)

Attributs	Méthodes
id : un entier identifiant un composant de manière unique. associée au composant.	**getHelp :** renvoie l'aide pour le joueur

Nom de la classe : Card (sous-type de Component)

Attributs	Méthodes
value : un entier représentant la valeur de la carte.	

Nom de la classe : Token (sous-type de Component)

Attributs	Méthodes
type : une chaîne de caractères décrivant le type de jetons (type dépendant du jeu).	

Nom de la classe : Pawn (sous-type de Component)

Attributs	Méthodes
color : une chaîne de caractères qui définit la couleur du pion.	

Nom de la classe : Tile

Attributs	Méthodes
type : une chaîne de caractères décrivant le type de tuile (type dépendant du jeu).	

Nom de la classe : Game

Attributs	Méthodes
id : un entier identifiant une partie de manière unique.	
launched : un booléen indiquant si la partie est lancée.	

6 Architecture dynamique

6.1 GameController

GameController désigne ici n'importe quel contrôleur de jeu. Game peut donc être remplacé par SixQP, Splendor, Glenmore, Myrmes.

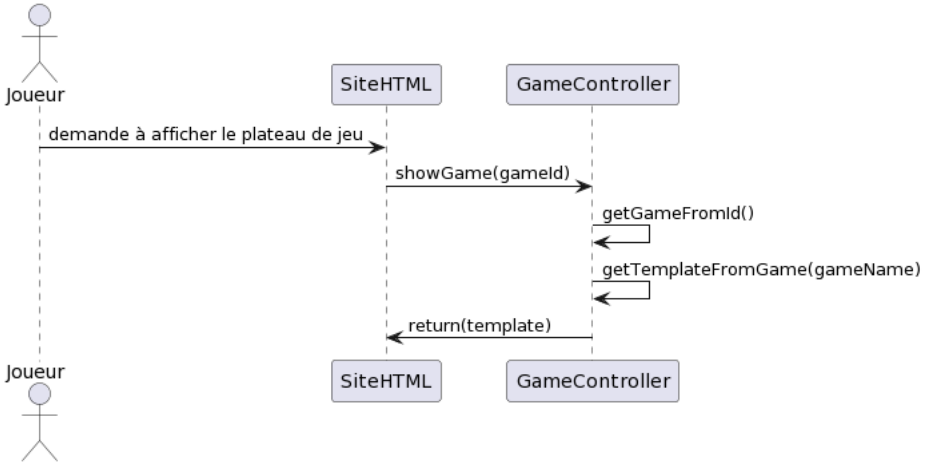

FIGURE 5 – Diagramme de séquence de l'affichage du plateau d'un jeu (STB-INT-04)

L'objet HubInterface est un objet fourni par la bibliothèque de code de Mercure

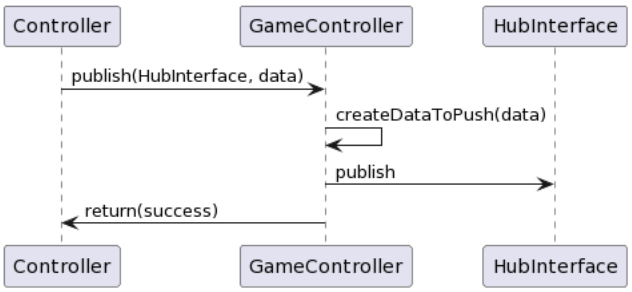

FIGURE 6 – Diagramme de séquence de la publication aux navigateurs des mises à jour du plateau (STB-INT-01)

6.2 MessageController

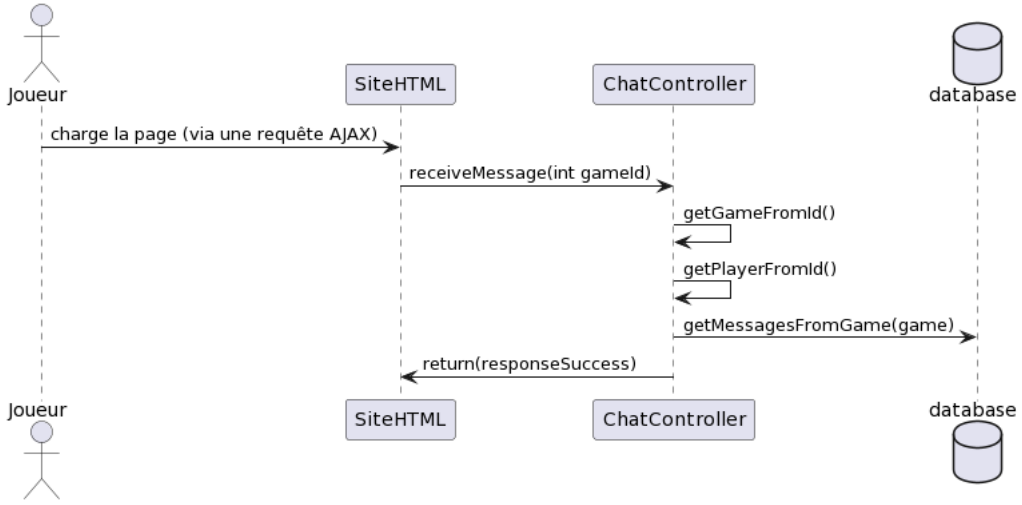

FIGURE 7 – Diagramme de séquence de la réception des messages (STB-GES-03)

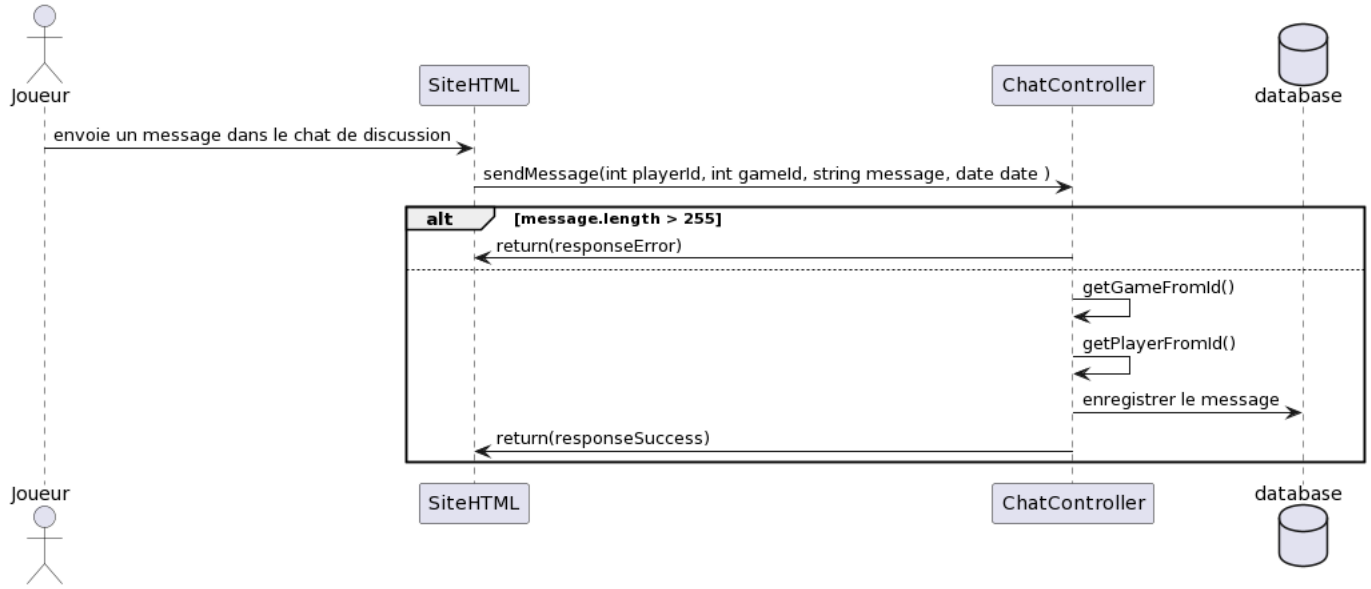

FIGURE 8 – Diagramme de séquence de l'envoi d'un message (STB-GES-03)

6.3 AbstractGameManagerService

GameManagerService appelle selon ses paramètres XGameManagerService par l'intermédiaire de AbstractGameManagerService.

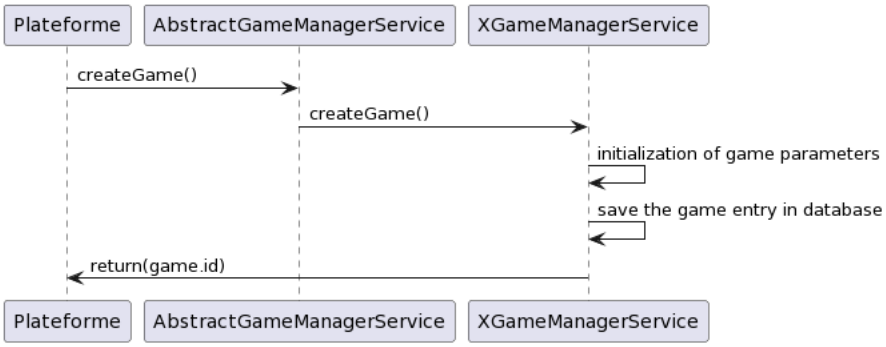

FIGURE 9 – Diagramme de séquence de la création d'une partie

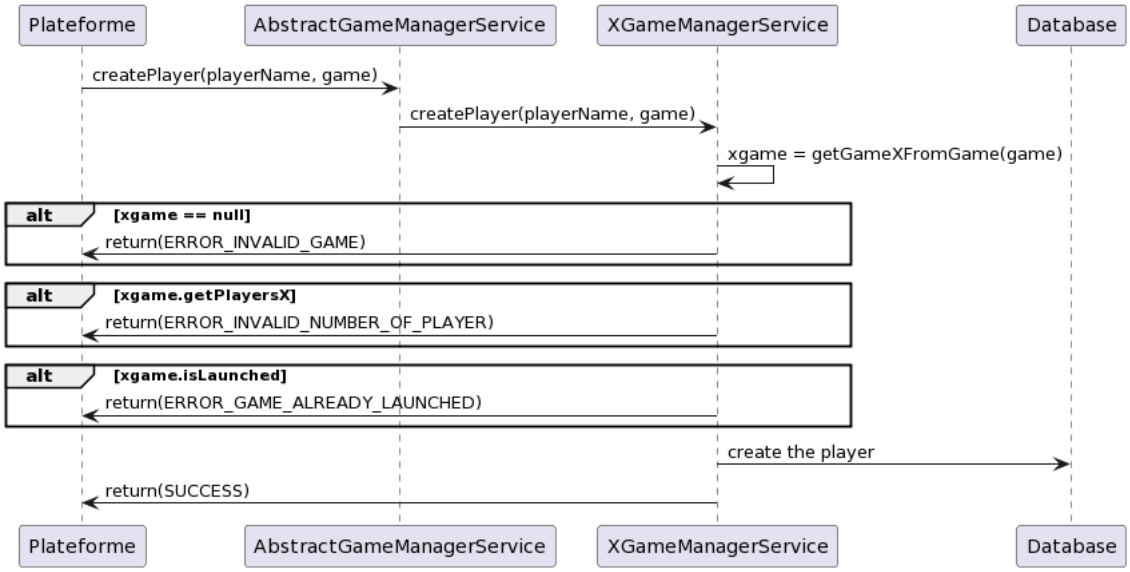

FIGURE 10 – Diagramme de séquence de l'ajout d'un joueur à une partie

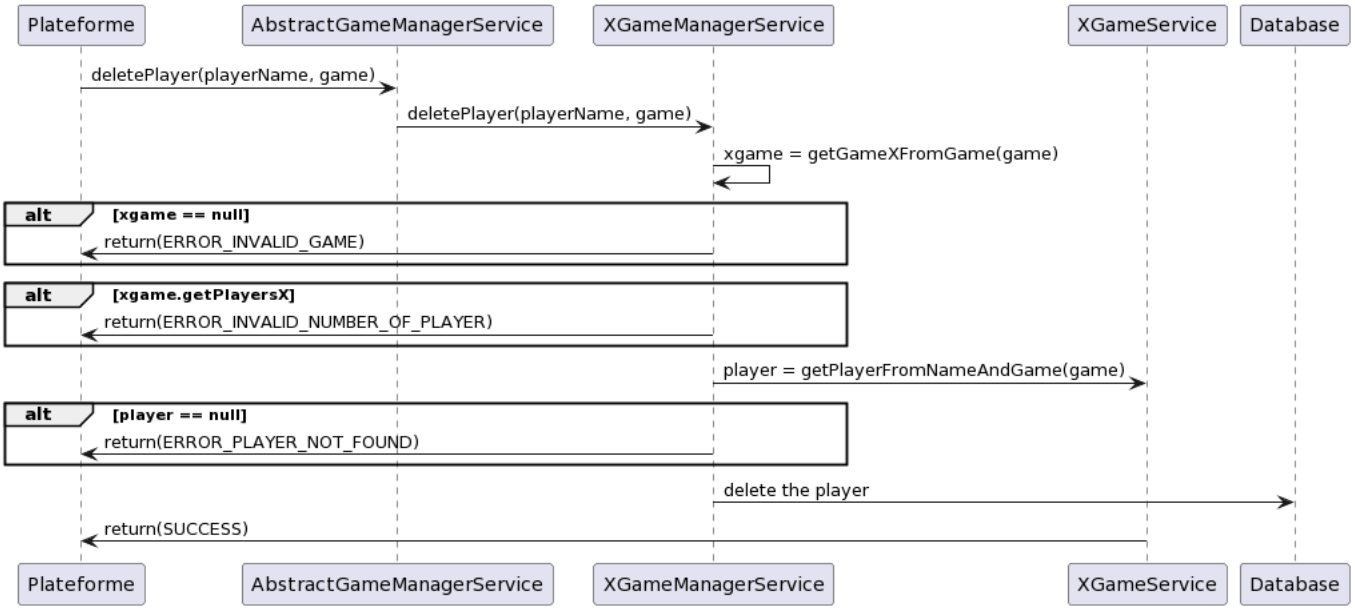

FIGURE 11 – Diagramme de séquence de la suppression d'un joueur d'une partie

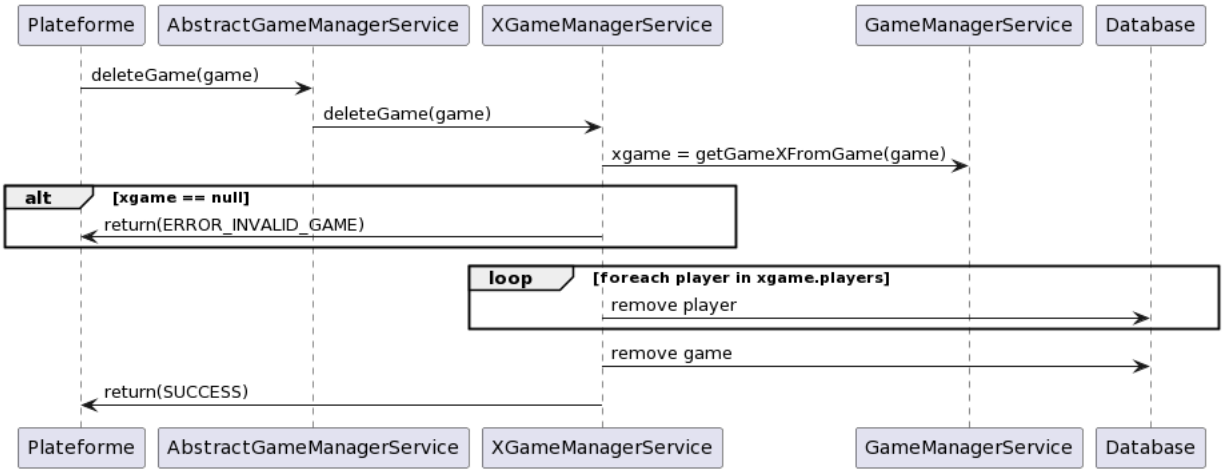

FIGURE 12 – Diagramme de séquence de la suppression d'une partie

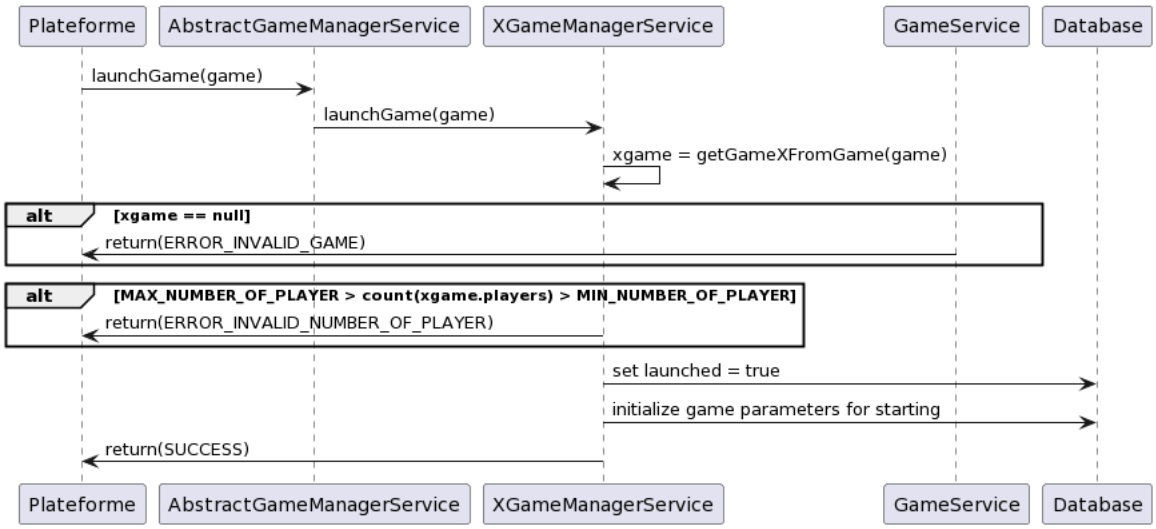

FIGURE 13 – Diagramme de séquence du lancement d'une partie