Document d'architecture logicielle

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
aaaa-mm-jj	x.x	<Détails précis du travail effectué>	<Nom>
2023-03-09	1.0	Ajout des diagrammes/texte pour tout sauf vue déploiement	Maxime Aspros
2023-03-17	1.1	Vue logique: réorganisation des paquetages en fonction de l'architecture. Vue de déploiement: Ajout du diagramme.	Maxime Aspros
2023-03-21	1.2	Vue logique: mise à jour de certains diagrammes, ajout de diagramme de paquetages pour fonctionnalités du Sprint 3 Ajout des titres pour les diagrammes, retrait des instructions en bleu	Maxime Aspros
2023-04-19	2.0	Révision du document en fonction des rétroactions du Sprint 2	Maxime Aspros

Table des matières

1. Introduction 4

2. Vue des cas d'utilisation 4

3. Vue des processus 11

4. Vue logique 16

5. Vue de déploiement 25

Document d'architecture logicielle

1. Introduction

L’intention de ce document est d’illustrer l’architecture logicielle utilisée par l'équipe 203 dans la réalisation du projet 2. Afin de mieux représenter l'implémentation des fonctionnalités requises, il contient la vue des cas d’utilisation, la vue des processus, la vue logique, ainsi que la vue de déploiement. Bien que les requis modélisés concernent principalement le Sprint 3, on retrouve également des fonctionnalités pertinentes des Sprints 1 et 2 pour faciliter la compréhension.

2. Vue des cas d’utilisation

Les requis du Sprint 3 imposent la conception de nouvelles fonctionnalités qui amènent des cas d’utilisation originaux, telles que le mode Temps Limité ainsi que le mode Reprise Vidéo.

L'implémentation du Sprint 3 contient également plusieurs fonctionnalités complémentaires venant s’ajouter à celles implémentées lors des remises précédentes. On note par exemple l'implémentation des indices de jeu, la configuration des constantes de jeu, ou bien l’historique des parties jouées entre autres.

L'utilité des diagrammes de cas d’utilisation est de visualiser à haute résolution les fonctionnalités ainsi que les interactions possibles entre l’utilisateur et le client. Les cas d’utilisation sont ensuite individuellement détaillés en illustrant la séquence d’évènements qui les composent.

Diagramme de contexte de tous les cas d'utilisation

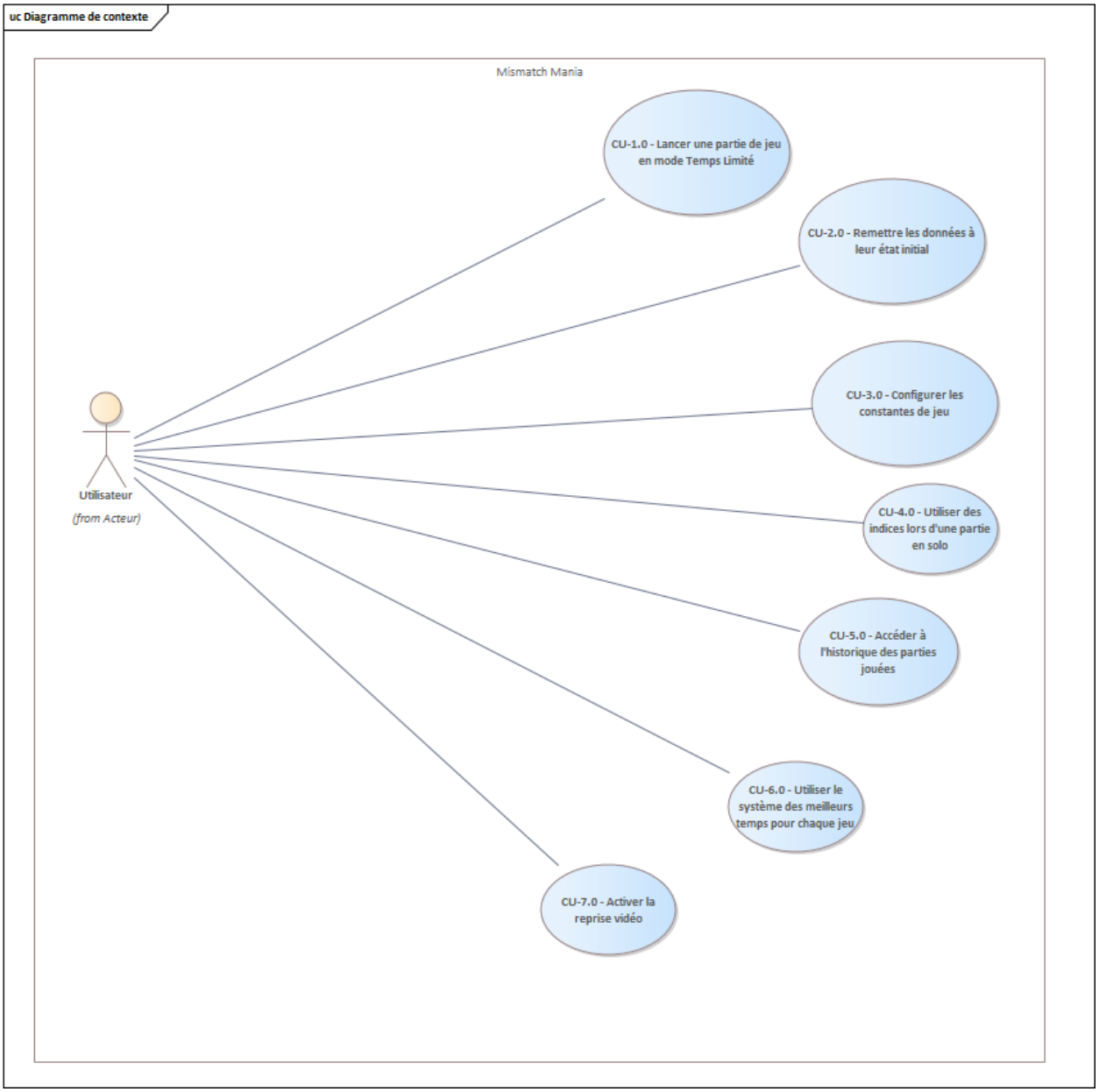


Diagramme de contexte CU1.0 - Lancer une partie de jeu en mode Temps Limité

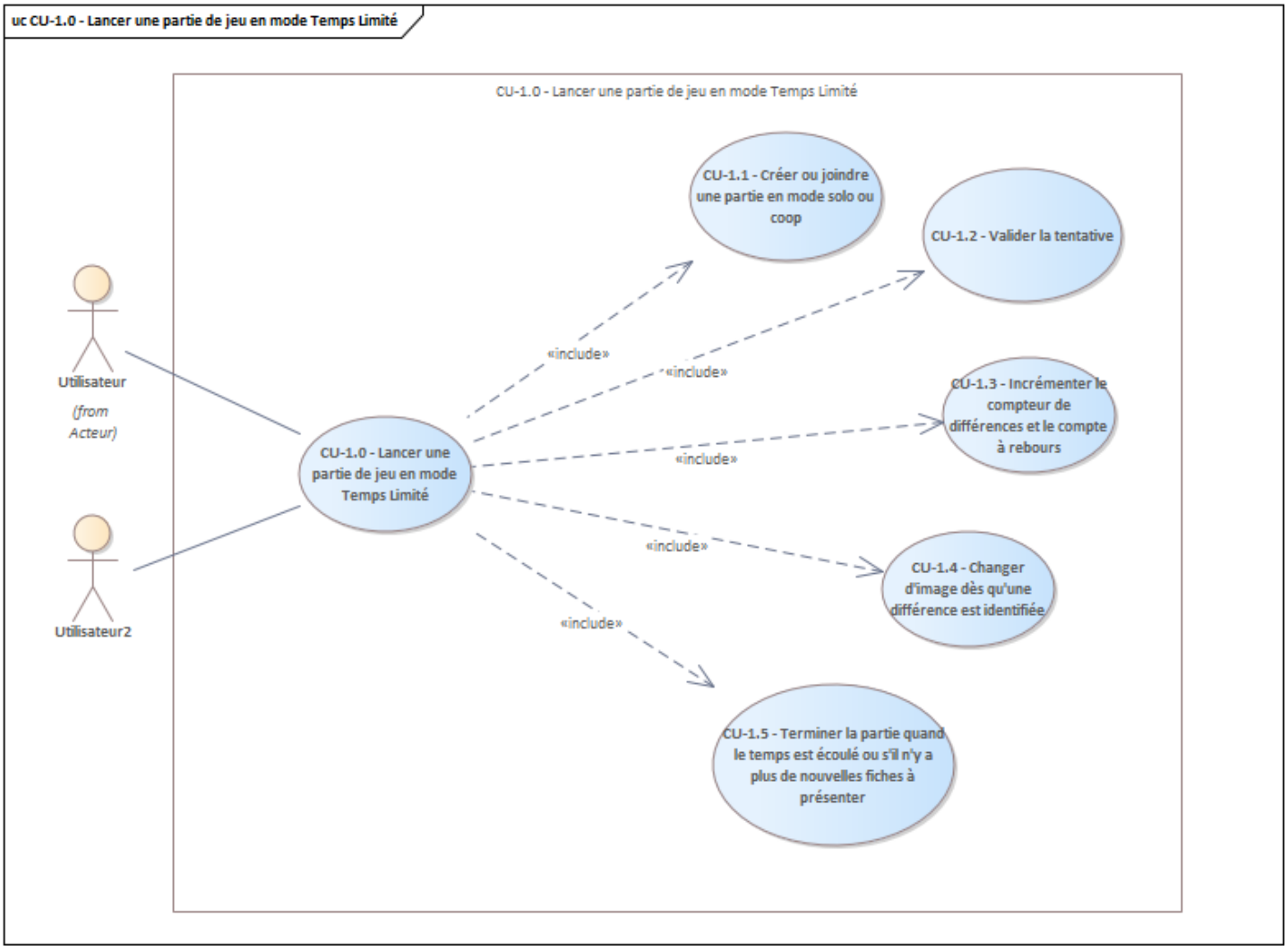


Diagramme de contexte CU2.0 - Remettre les données à leur état initial

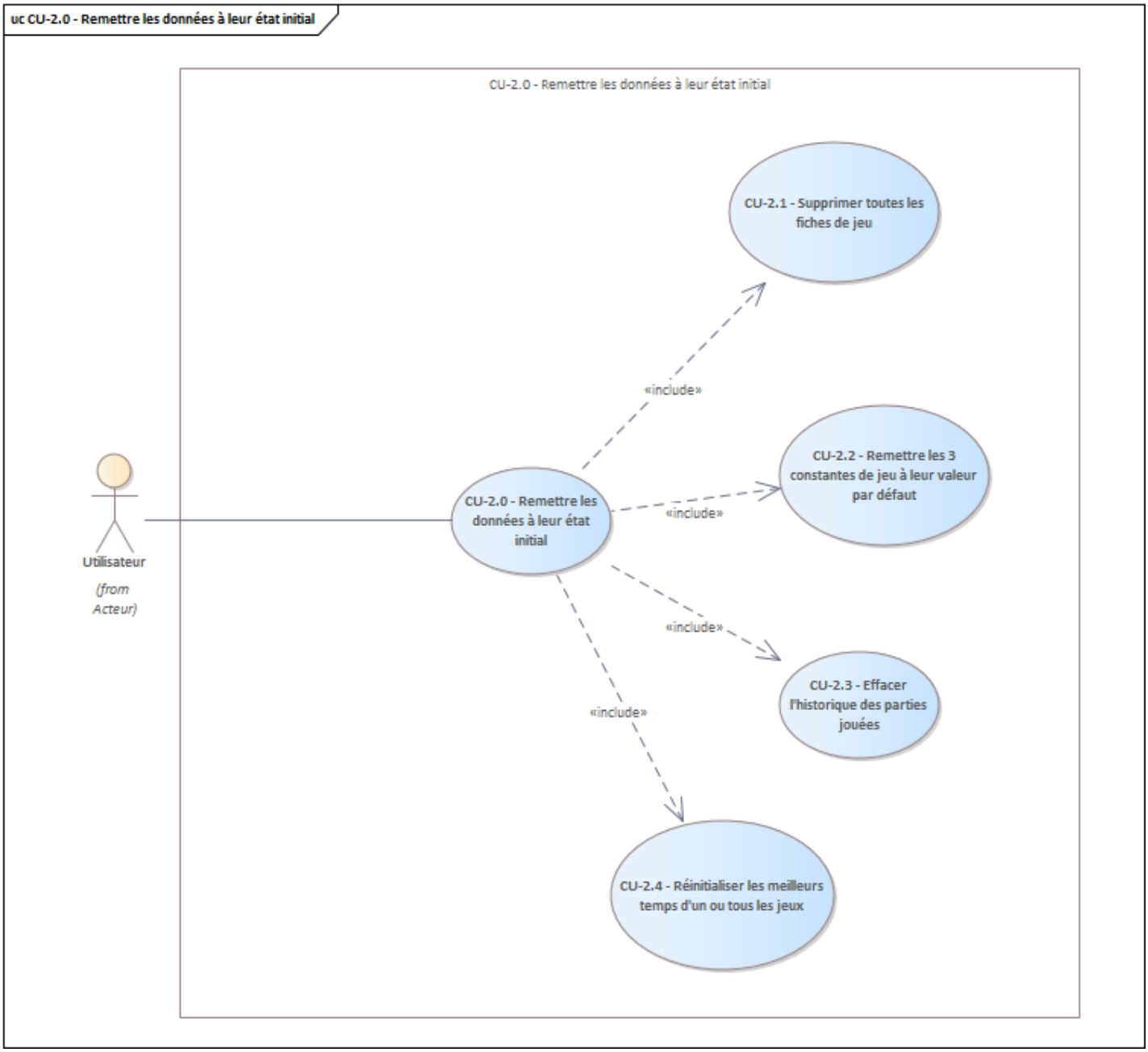


Diagramme de contexte CU3.0 - Configurer les constantes de jeu

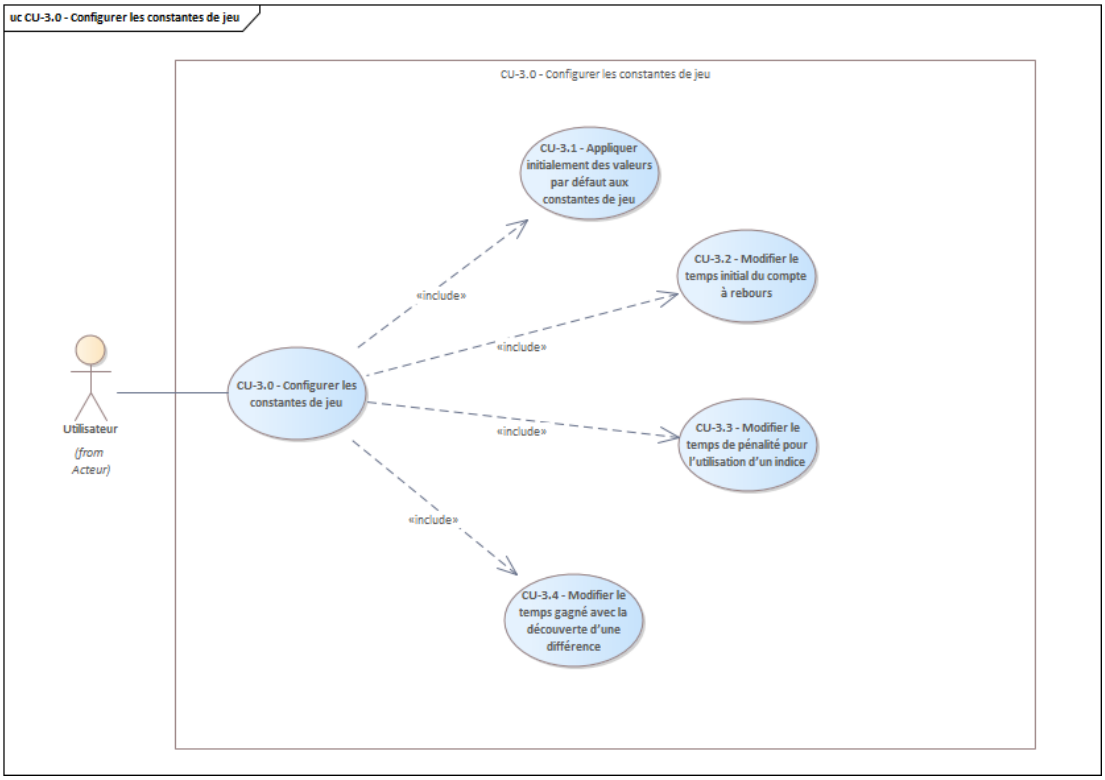


Diagramme de contexte CU4.0 - Utiliser des indices lors d'une partie en solo

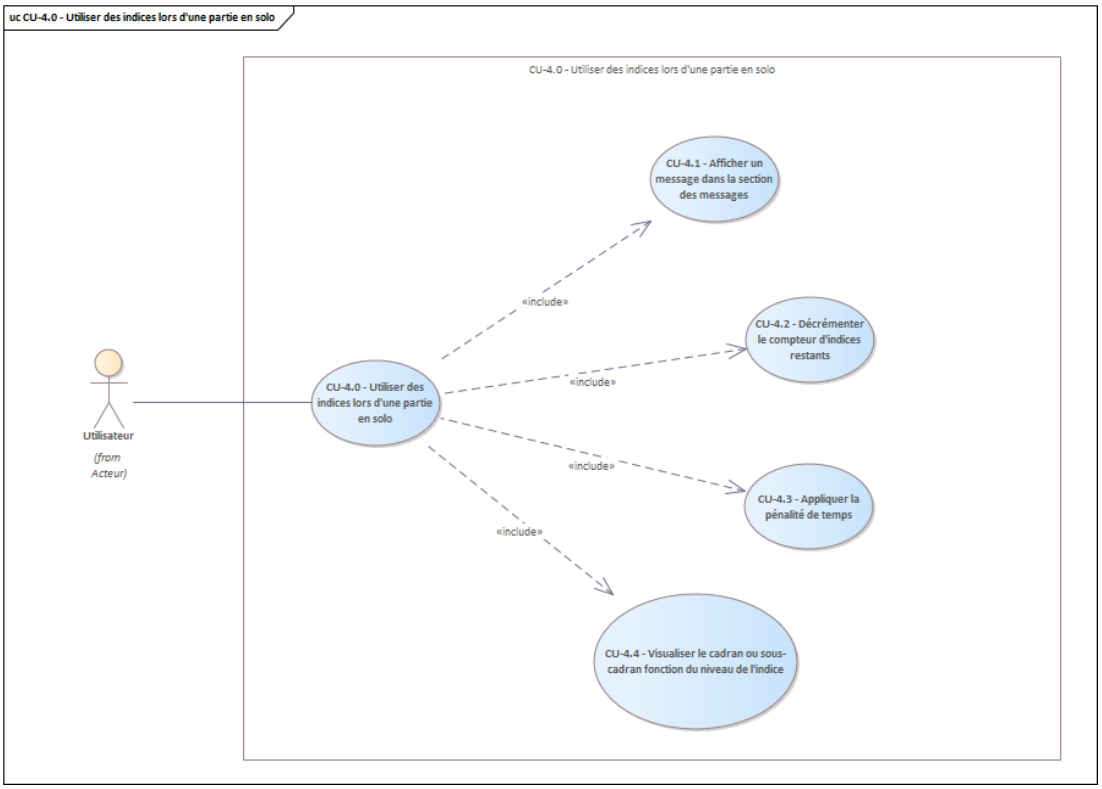


Diagramme de contexte CU5.0 - Accéder à l'historique des parties jouées

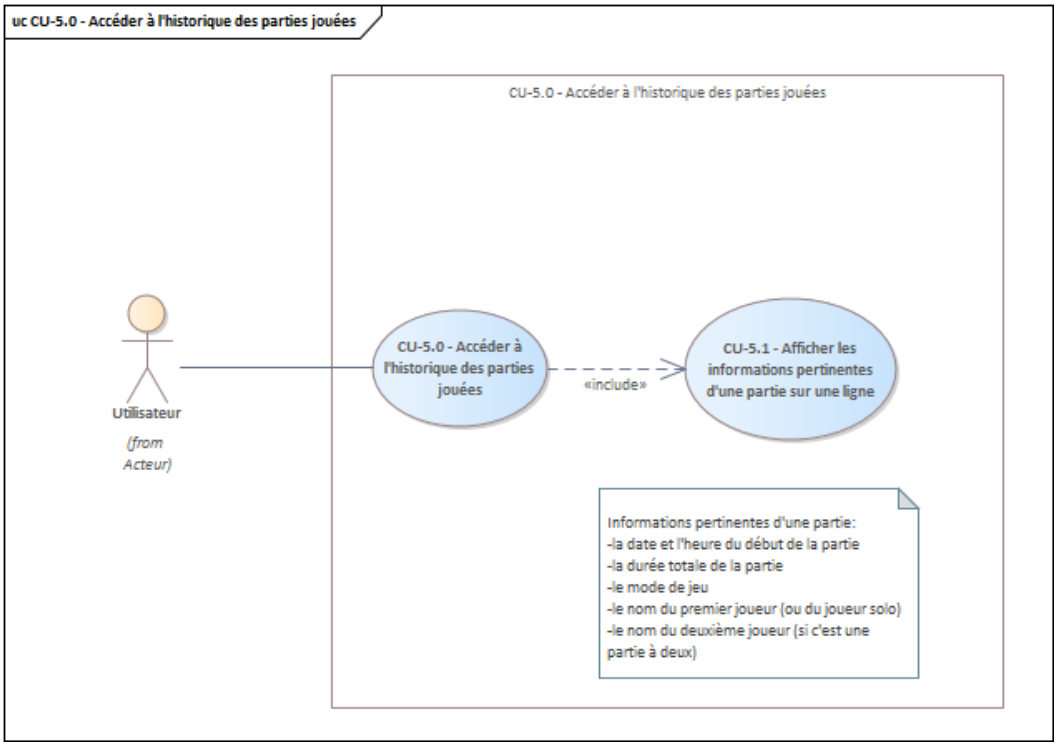


Diagramme de contexte CU6.0 - Utiliser le système des meilleurs temps pour chaque jeu

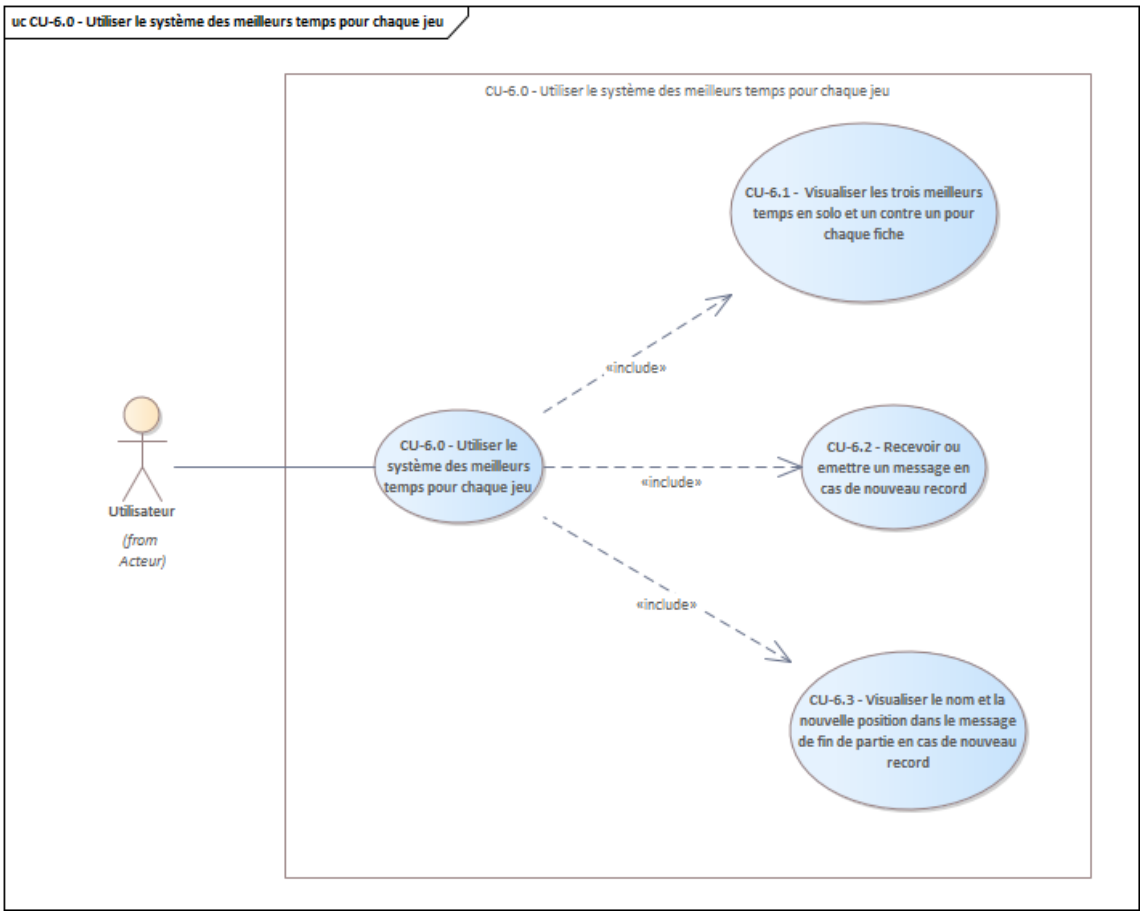
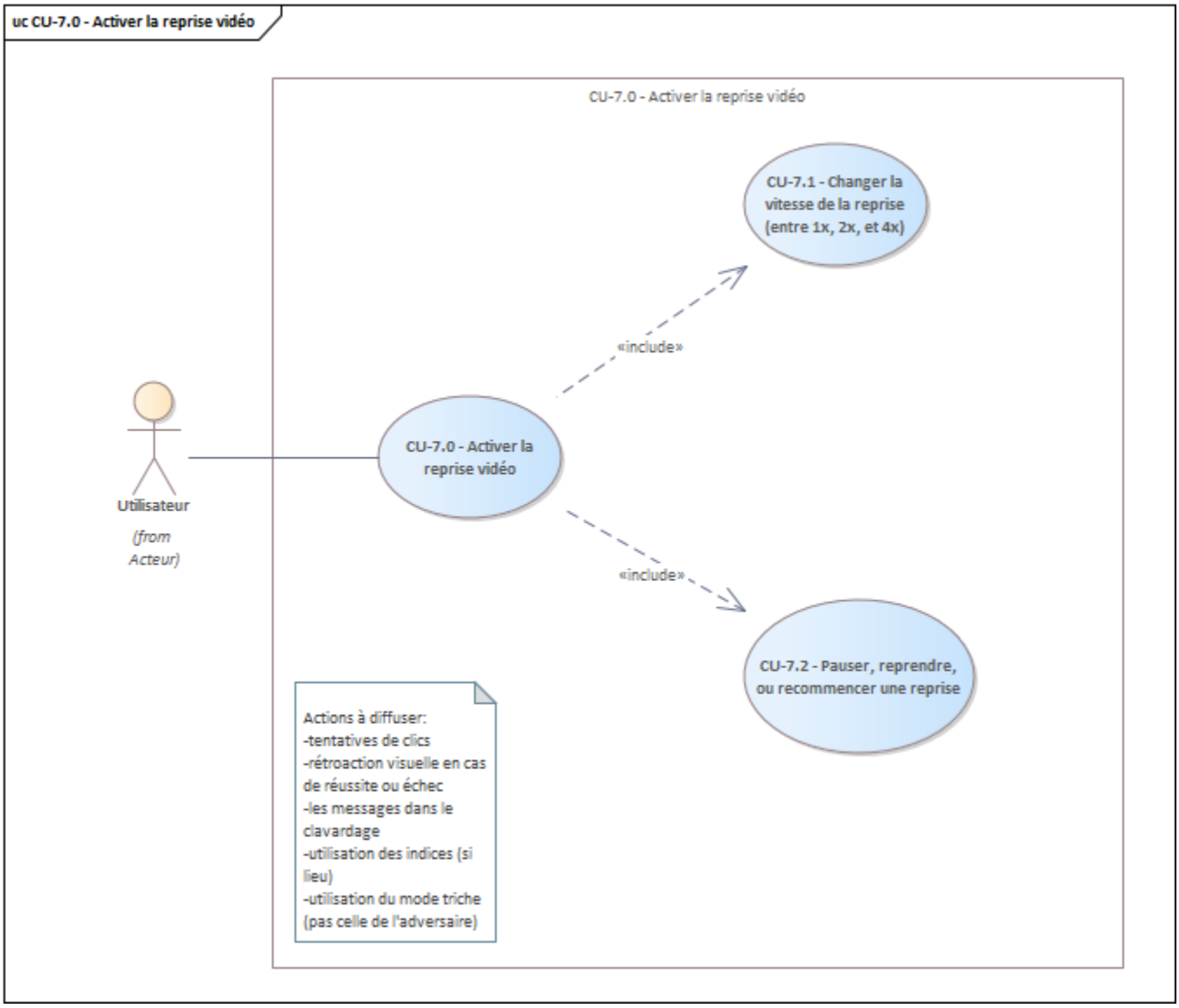


Diagramme de contexte CU7.0 - Activer la reprise vidéo



3. Vue des processus

Diagramme d'interaction - Mode Temps Limité

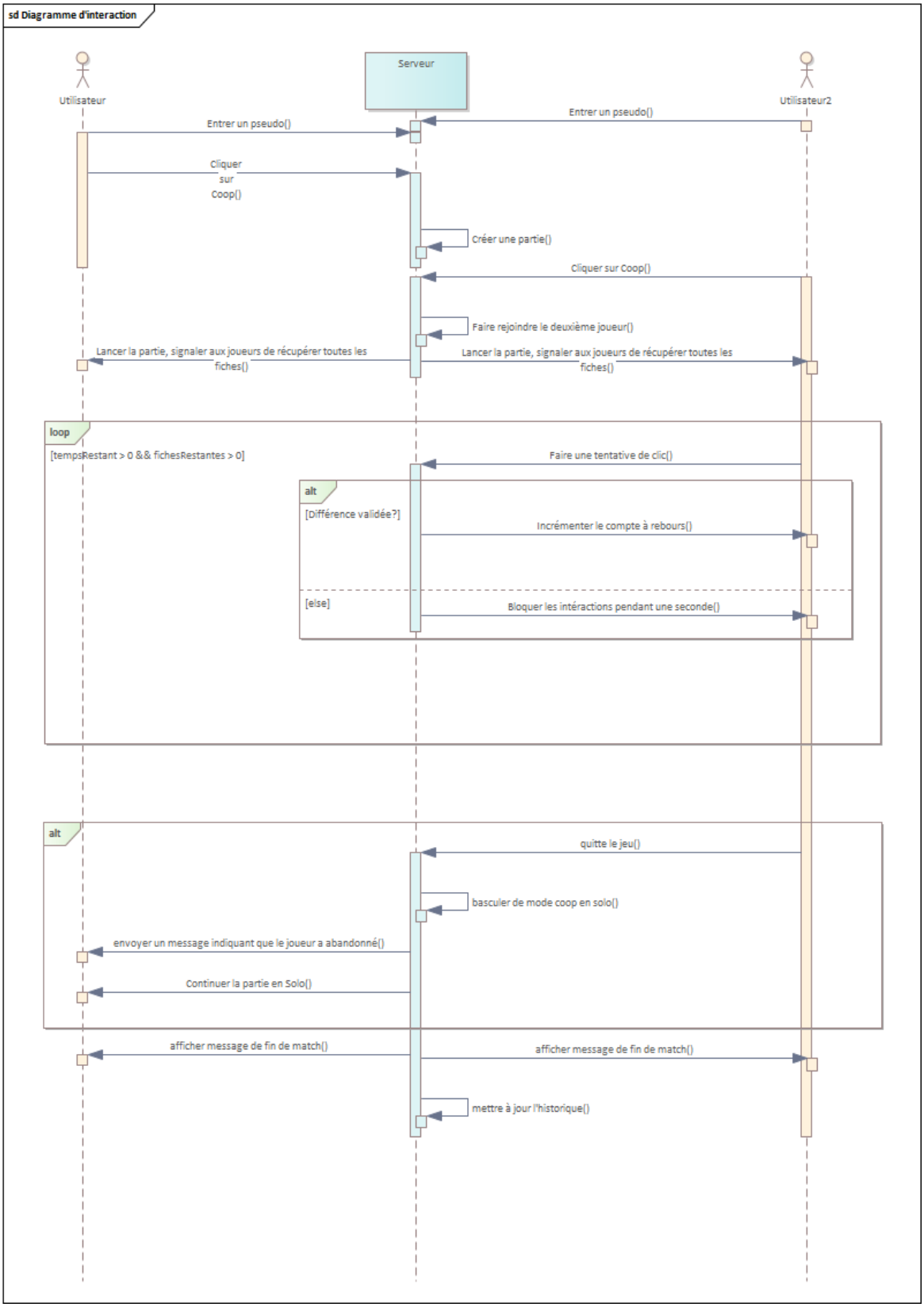


Diagramme d'interaction - Meilleurs temps

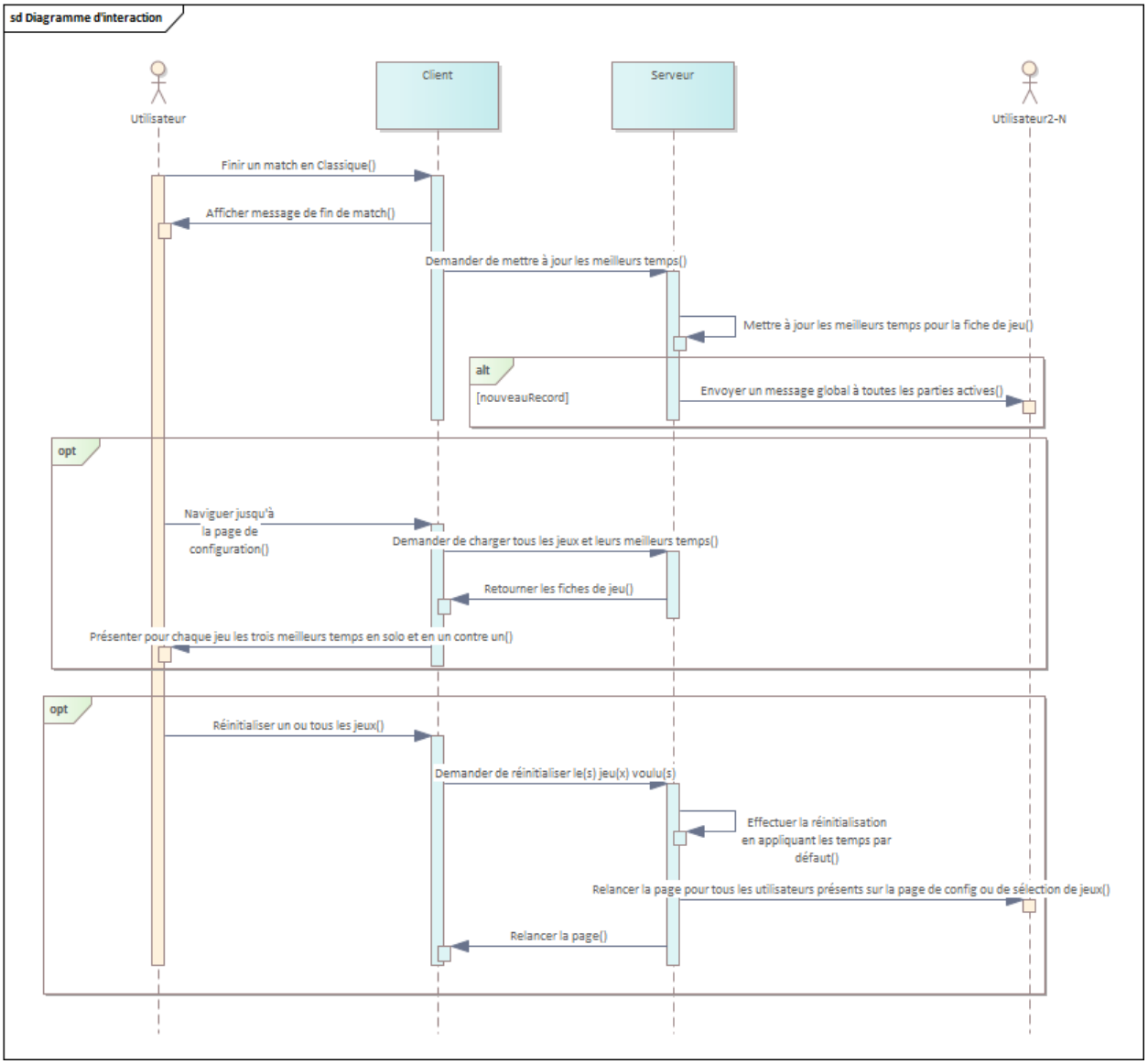


Diagramme d’interaction - Constantes de jeu et historique

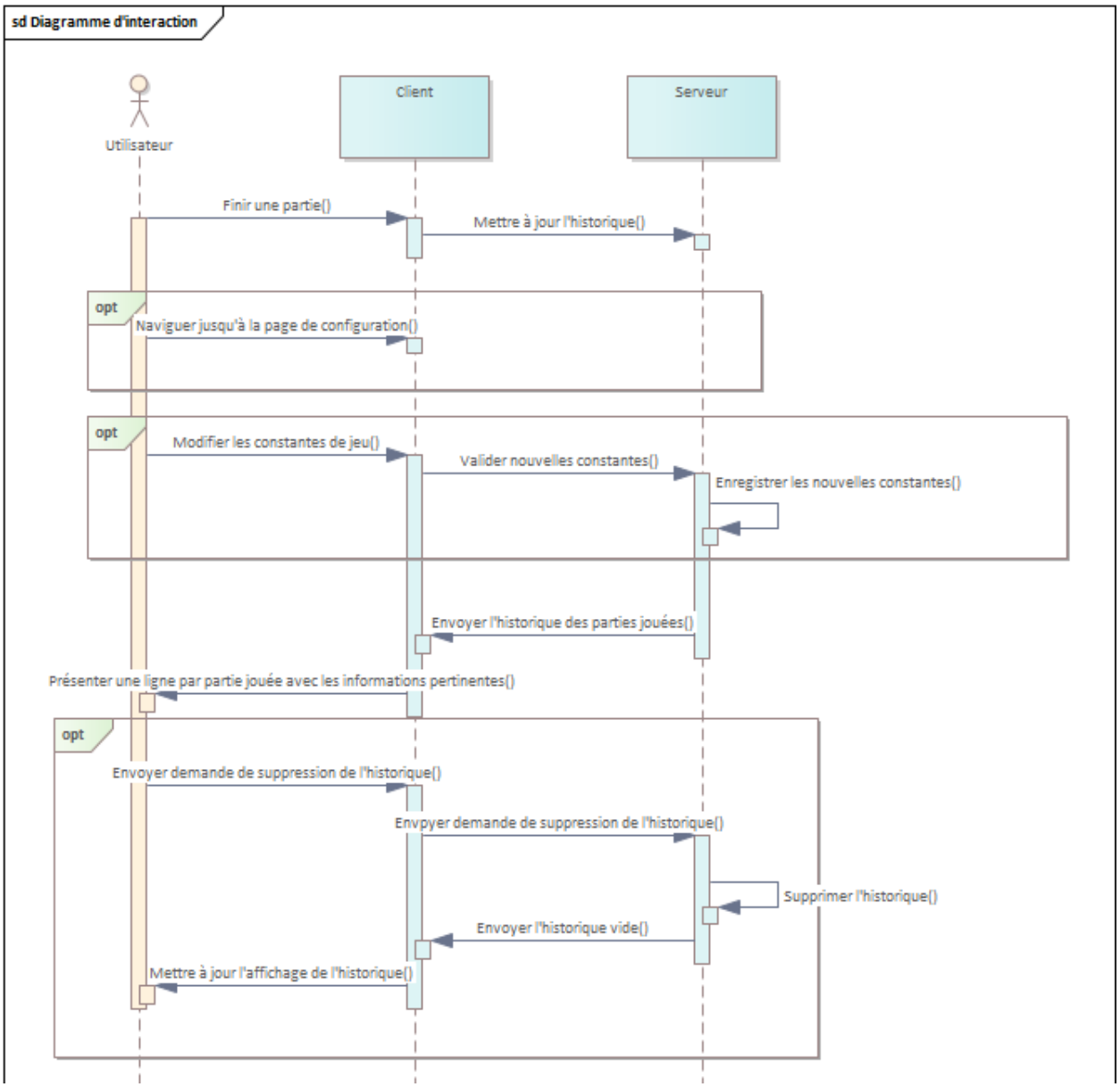


Diagramme d'interaction - Mode Reprise

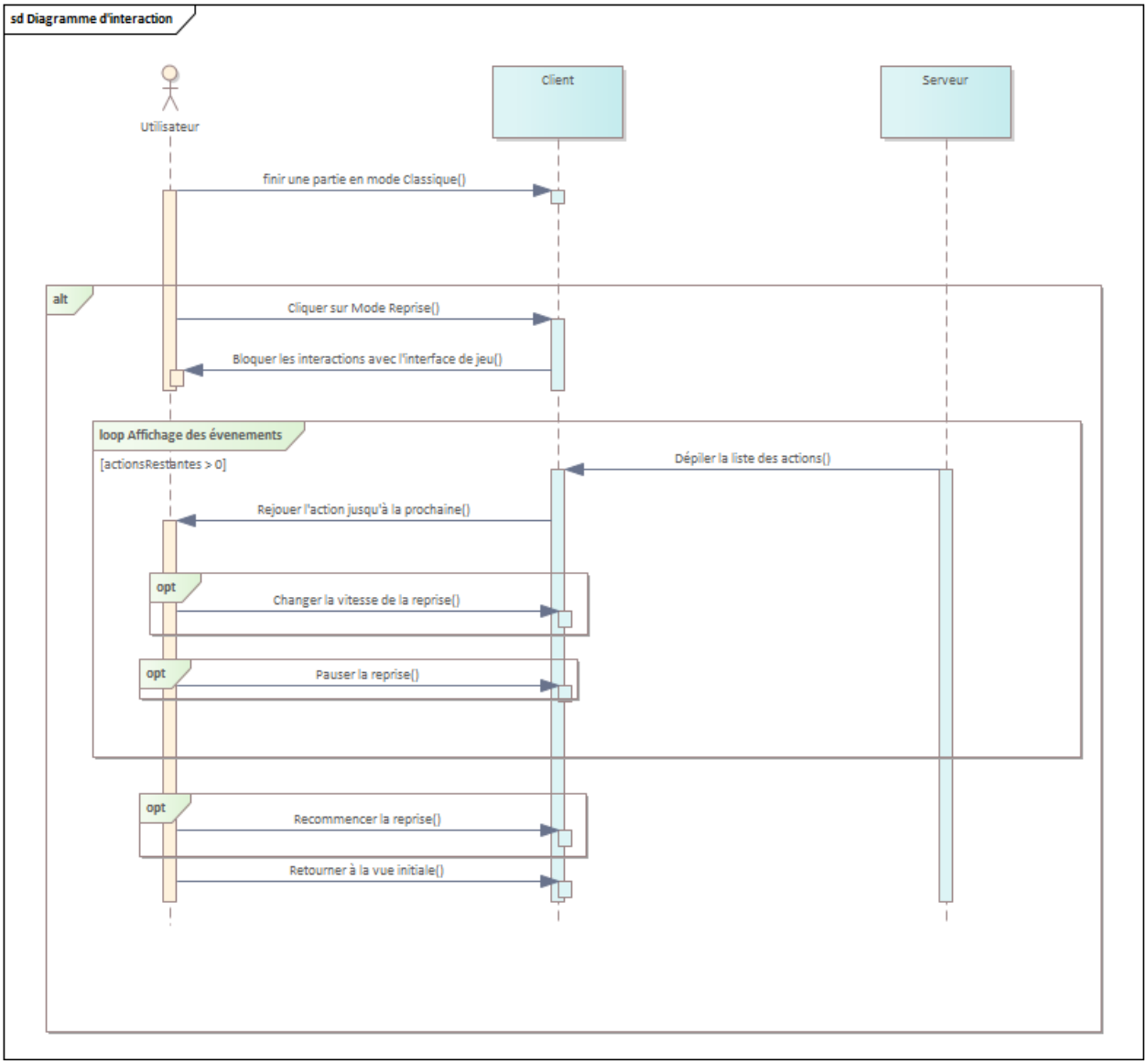
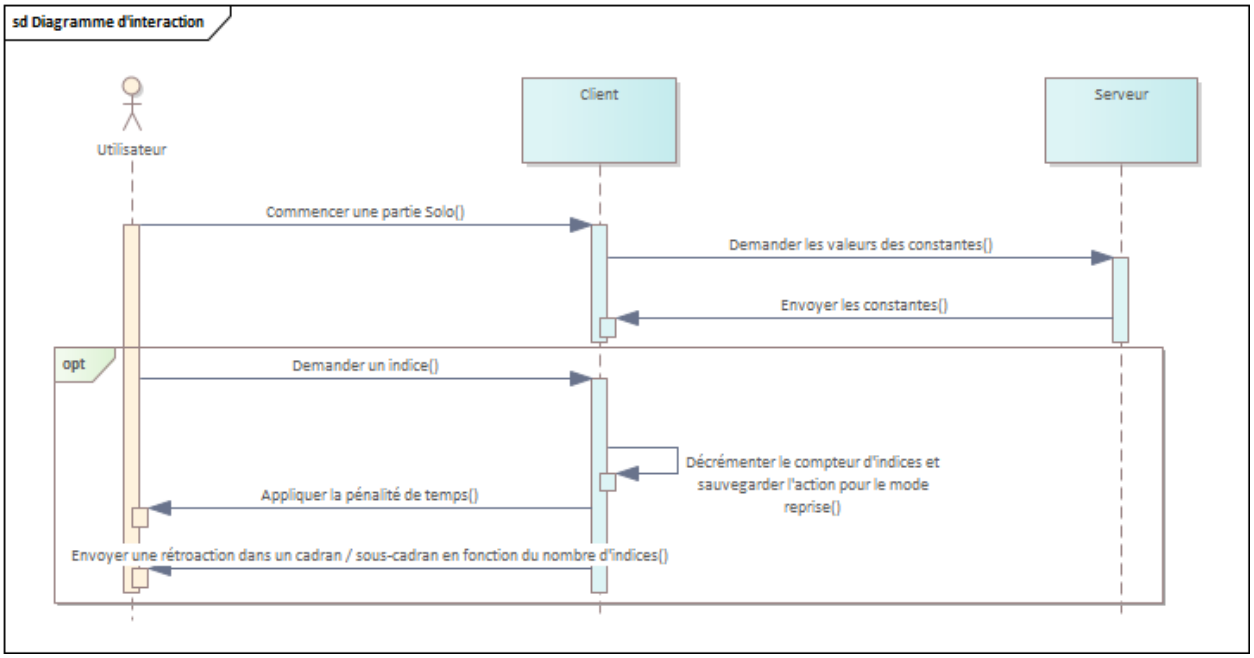


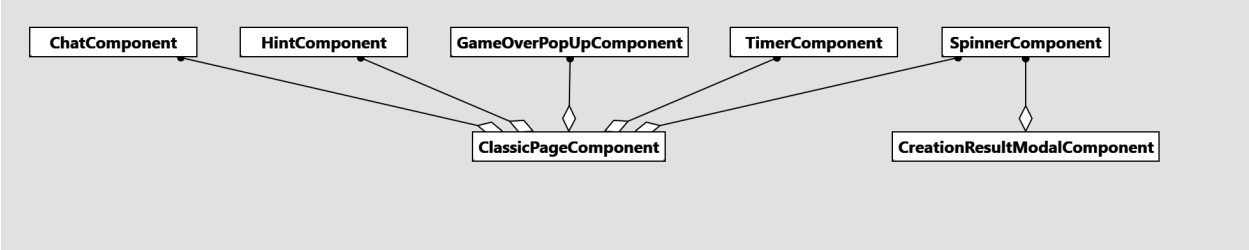
Diagramme d’interaction - Utilisation des indices



4. Vue logique

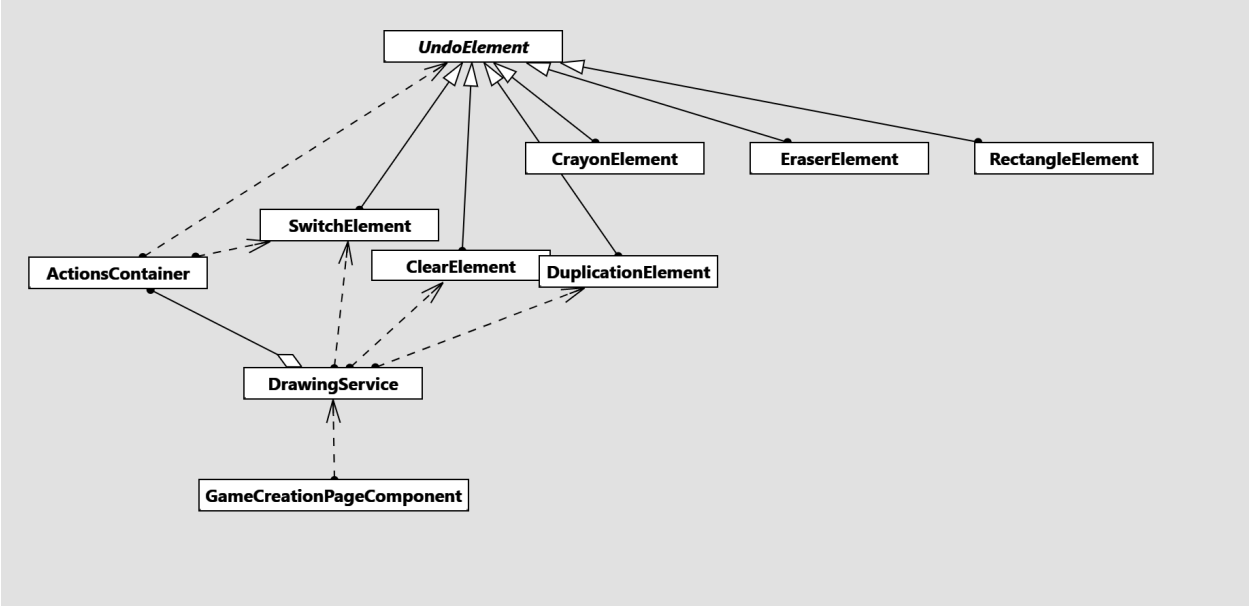
Client: Vue
La majorité du code relié à la vue se retrouve dans les composantes des pages, mais de nombreuses composantes auxiliaires aident à réduire la duplication de code et donner un rendu plus flexible et modulable. Pour la page Classique, on note l'utilisation de composantes individuelles pour le clavardoir, les *pop-ups*, les indices, le compteur des différences trouvées ainsi que l'affichage de la minuterie. La page de sélection ainsi que la page de configuration utilisent toutes les deux la composante 'GamesDisplay'. On utilise un booléen afin d'instancier la composante en fonction de la page désirée. Lorsque le client veut supprimer un ou tous les jeux, la page lance une pop-up afin de confirmer son choix.

Diagrammes de classe - Vue



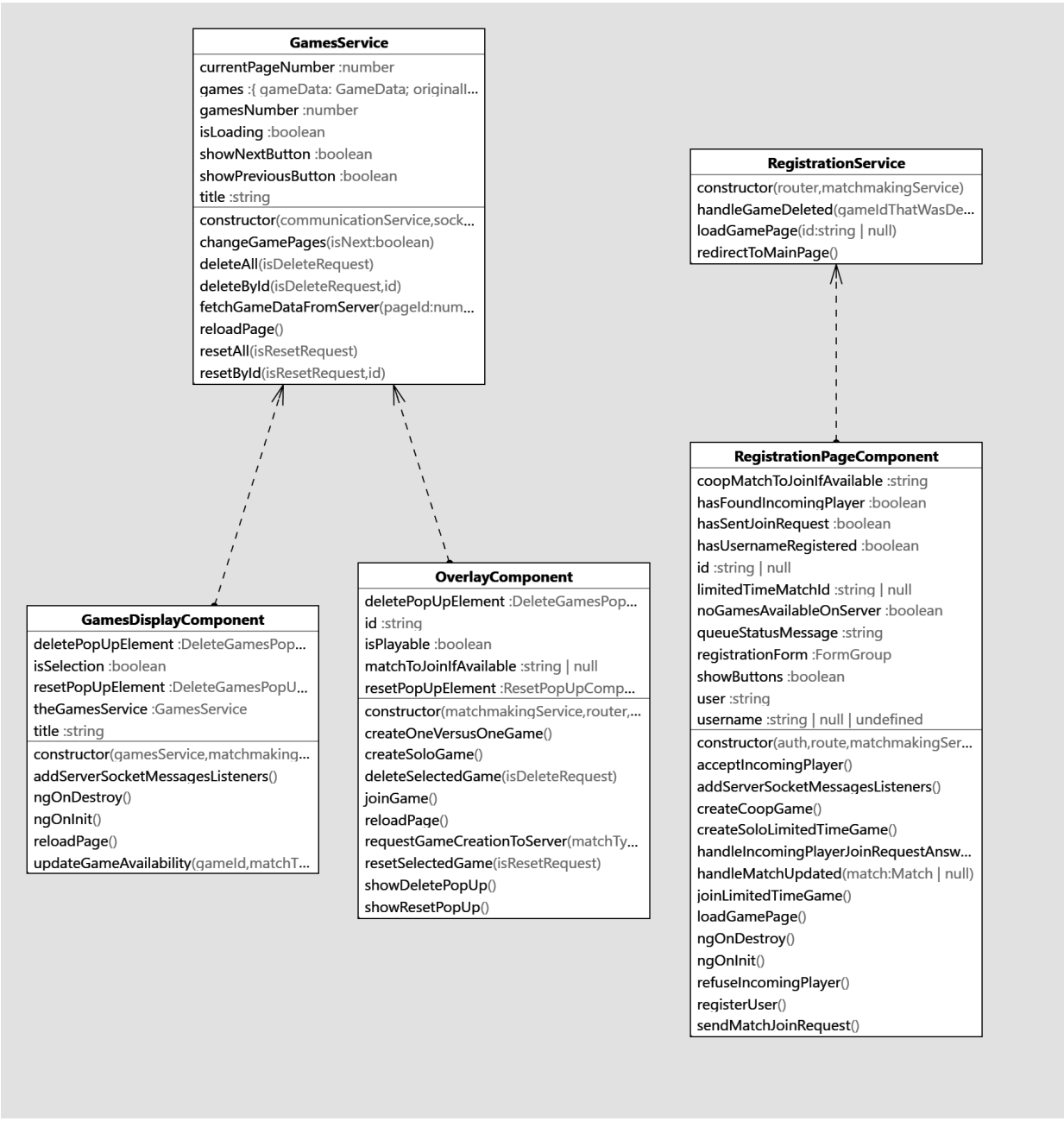
Client: Logique de modification de l'avant-plan
Le diagramme de classes ci-dessous illustre les classes pertinentes pour modifier l'avant-plan dans la page de création de jeu. L'utilisation du polymorphisme permet de regrouper toutes les actions effectuées dans un conteneur, et les refaire/défaire avec la méthode abstraite "applyElementAction" implémentée selon le type de sous-classe (crayon, rectangle, efface, etc…). Cela simplifie considérablement la logique d'ActionsContainer, puisque chaque outil possède sa propre implémentation. La liste des actions à défaire/refaire est dans la classe ActionsContainer, puisqu'elle a la responsabilité de gérer la fonctionnalité d'annuler ou refaire une action. Afin de découpler la responsabilité de modifier l'avant-plan de la page de création de jeu, on utilise DrawingService. On initialise ce service avec des références aux canvas de la page de création de jeu, ainsi que des références aux 3 outils possibles (crayon, rectangle, efface).

Diagrammes de classe - Logique de modification de l'avant-plan



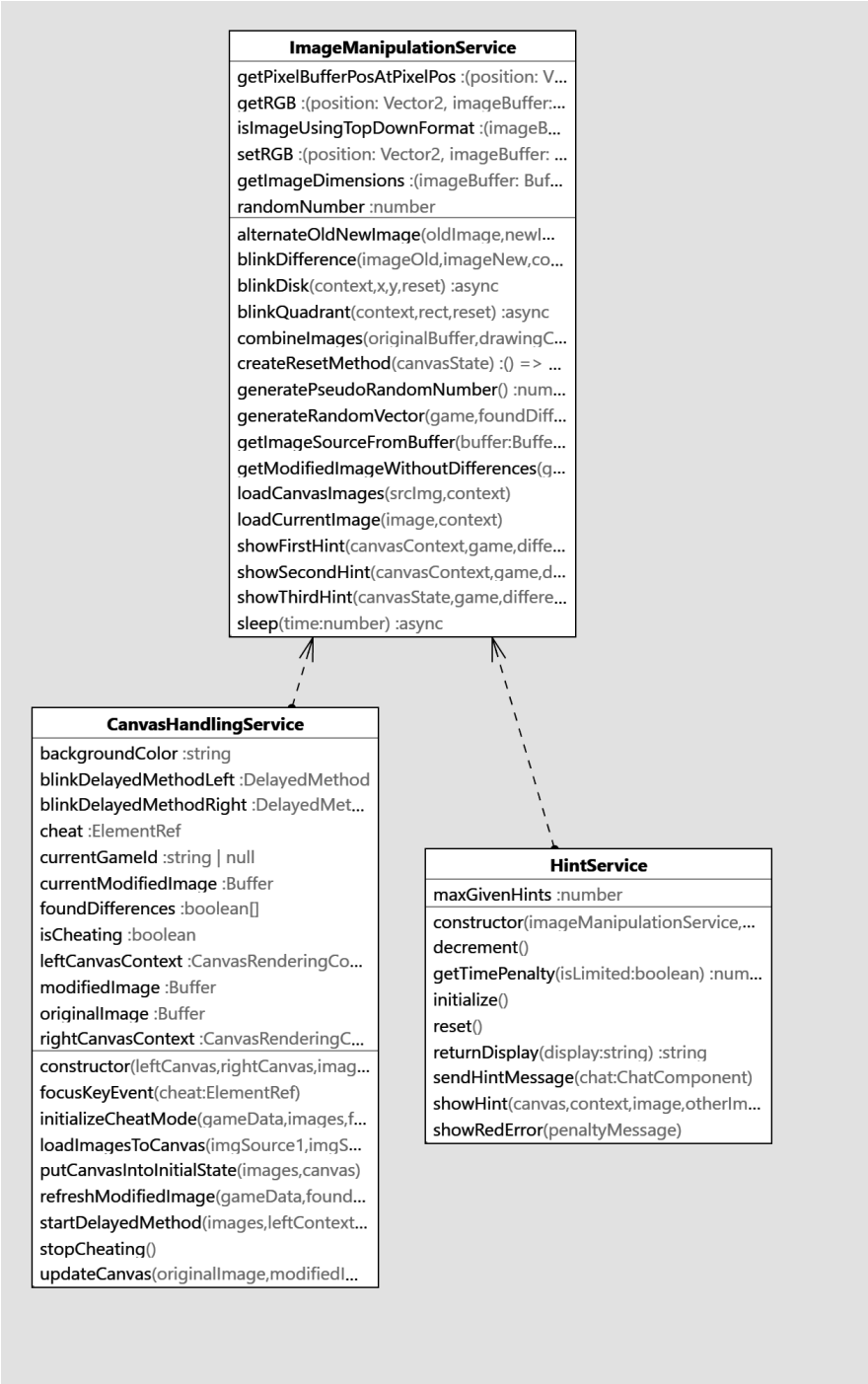
Client: Logique de création d'une partie
Créer et rejoindre une partie est un processus séquentiel qui commence d'abord par le page de sélection de jeu, en passant par la page d'enregistrement avant d'aboutir au jeu en tant que tel. GamesService récupère toutes les fiches et fournit l'identifiant de la salle à rejoindre (socketID) si un joueur est déjà en attente. Le mode Temps Limité reprend les mêmes étapes dans un ordre légèrement différent. On note par exemple que la création d'une salle de jeu s'effectue directement dans la page d'enregistrement, au lieu de la page de sélection de jeu.

Diagramme de classe - Logique de création d'une partie (client)



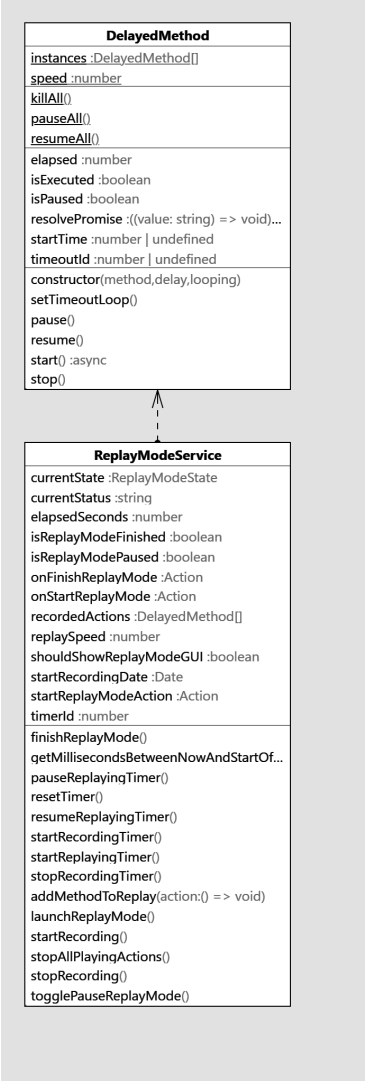
Client: Logique de la gestion des canvas
Afin d'alléger la page de jeu, la responsabilité de modifier les images est attribuée au CanvasHandlingService, qui est initialisé avec une référence aux deux canvas ainsi qu'au ImageManipulationService. Cela permet notamment d'activer le mode Triche. Les indices, quant à eux, sont gérés par leur propre service.

Diagramme de classe - Logique de la gestion des canvas



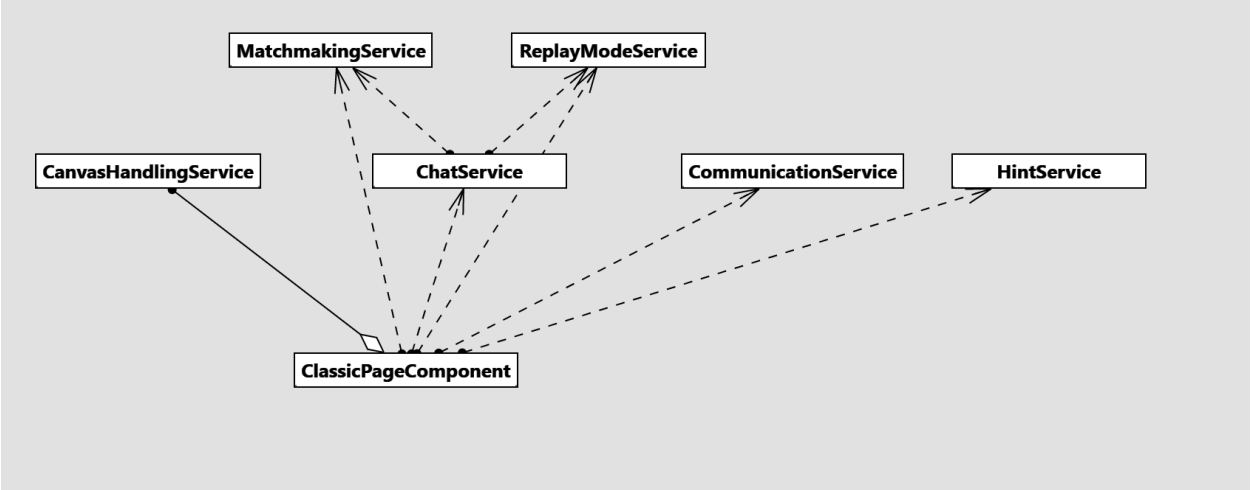
Client: Fonctionnement du Mode Reprise
La logique de contrôle du Mode Reprise est gérée par le service ReplayModeService. Tout au long du jeu, celui-ci empile des objets à la liste recordedActions, qui contient tous les événements effectués depuis le début du match. La liste recordedActions contient des éléments de la classe DelayedMethod afin de stocker plusieurs événements différents dans le même conteneur (ajout d'un message dans le clavardoir, détection d'une différence, utilisation du mode Triche, etc…) sous forme de fonction. ReplayModeService est initialisé au début du chargement de la page de jeu, puisque c'est celle-ci qui ajoute les actions effectuées à recordedActions. Cela permet de contourner le problème des actions spécifiques à un joueur, car chaque joueur possède sa propre instance des actions effectuées.

Diagramme de classe - Fonctionnement du Mode Reprise



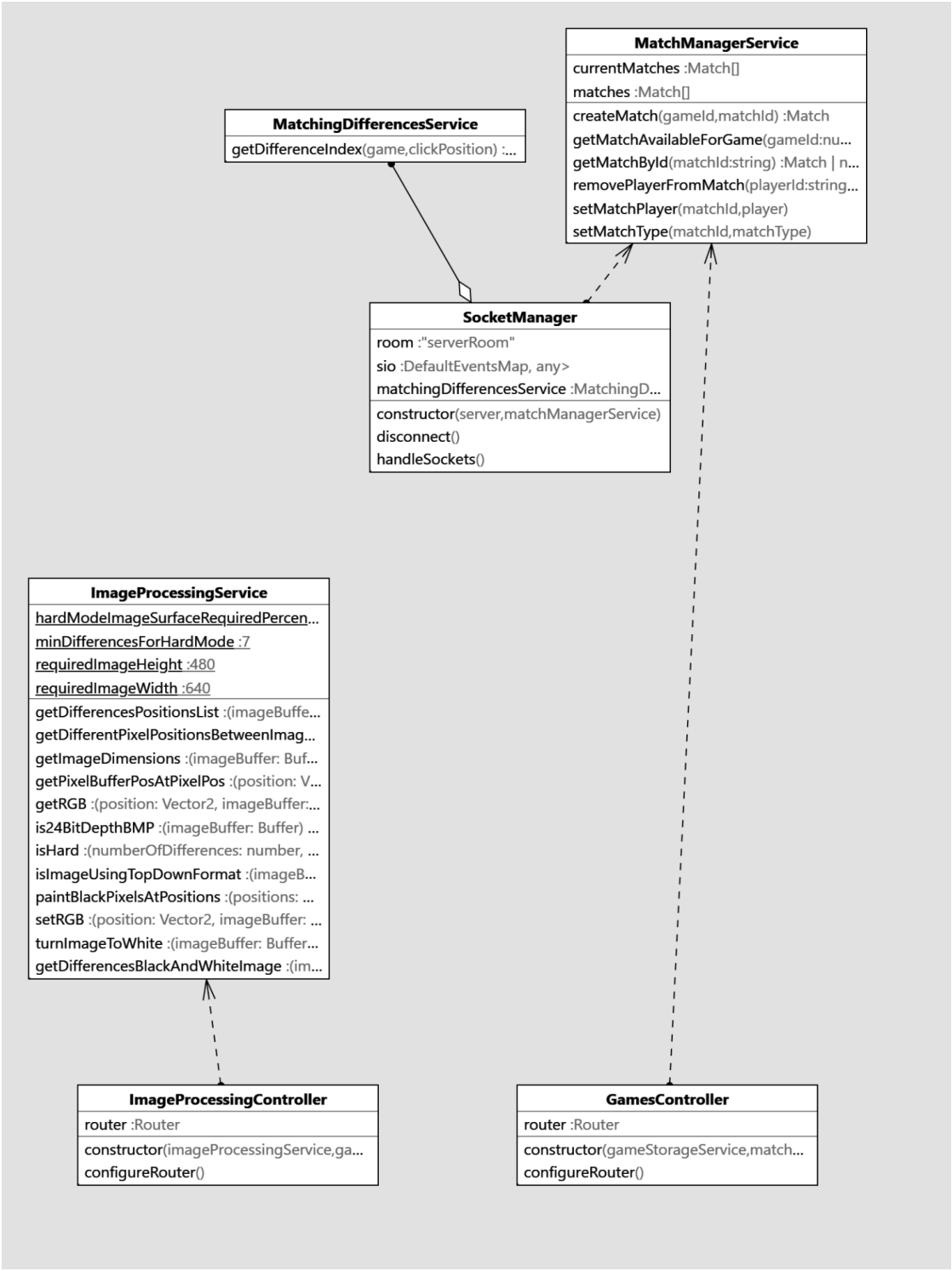
Client: Logique de fonctionnement d'une partie
Afin d'éviter la duplication de code, le mode Classique et Temps Limité sont intégrés dans la même page. Pour identifier une partie en Temps Limité, on met '-1' dans l'URL. S'il n'y a pas de fiches de jeu, une pop-up demande à l'utilisateur de créer au moins un jeu. Au chargement de la page de jeu, le client demande au serveur toutes les fiches de jeu disponibles. Ensuite, le compte à rebours est lancé et la partie est démarrée. En mode coopératif, la synchronisation est assurée en envoyant un événement Socket aux deux joueurs. On utilise également un événement Socket pour présenter les fiches aléatoirement (mais dans le même ordre) pour les deux joueurs. Si jamais l'un des deux joueurs quitte une partie en Coop, le service MatchmakingService s'occupe de changer le mode de jeu en Solo, ce qui modifie naturellement l'affichage HTML.

Diagramme de classe - Logique de fonctionnement d'une partie (client)



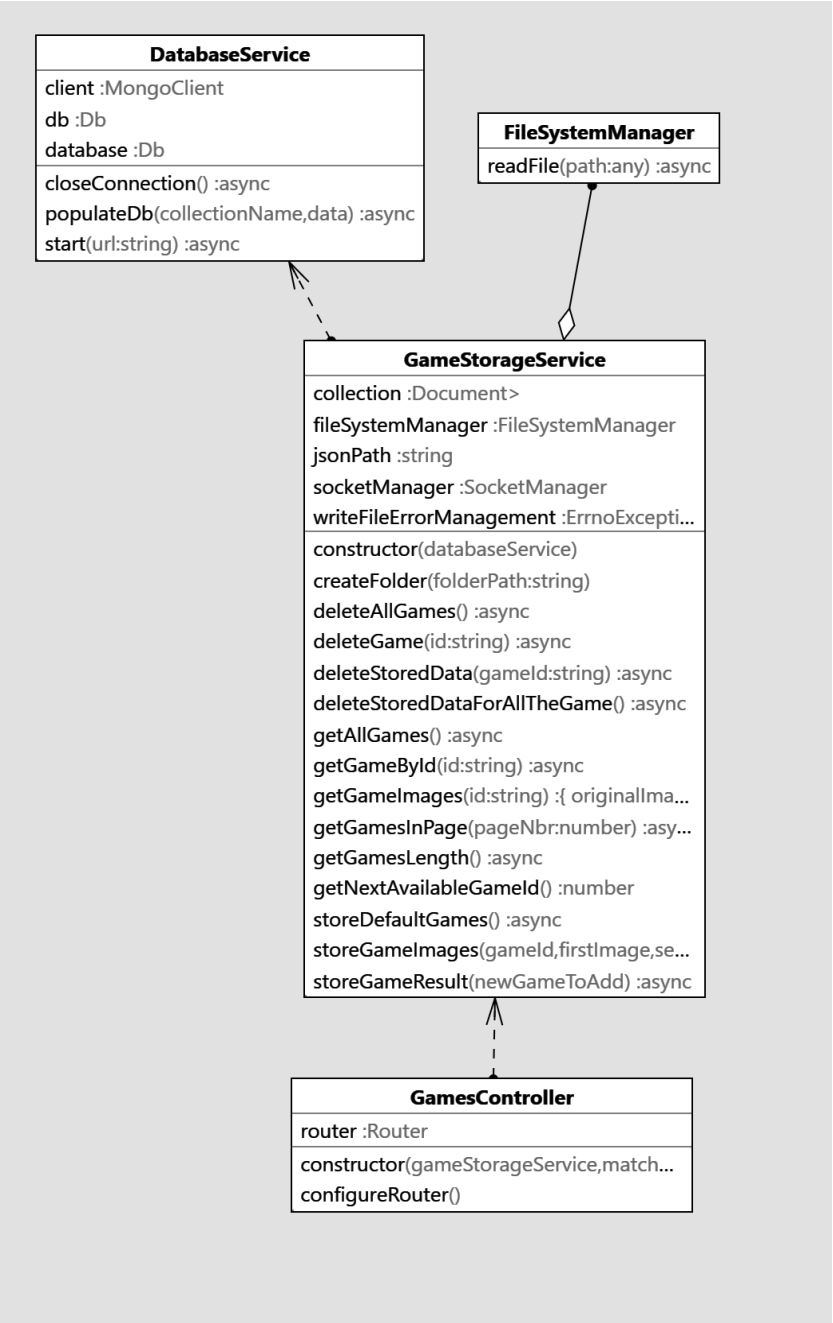
Serveur: Logique de fonctionnement d'une partie
Du côté serveur, les services MatchManagerService ainsi que SocketManagerService s'occupent de la réception et l'envoi des requêtes Socket.io lors d'une partie. Pour calculer si une tentative de clic est valide, on utilise également le MatchingDifferencesService.

Diagramme de classe - Logique de fonctionnement d'une partie (serveur)



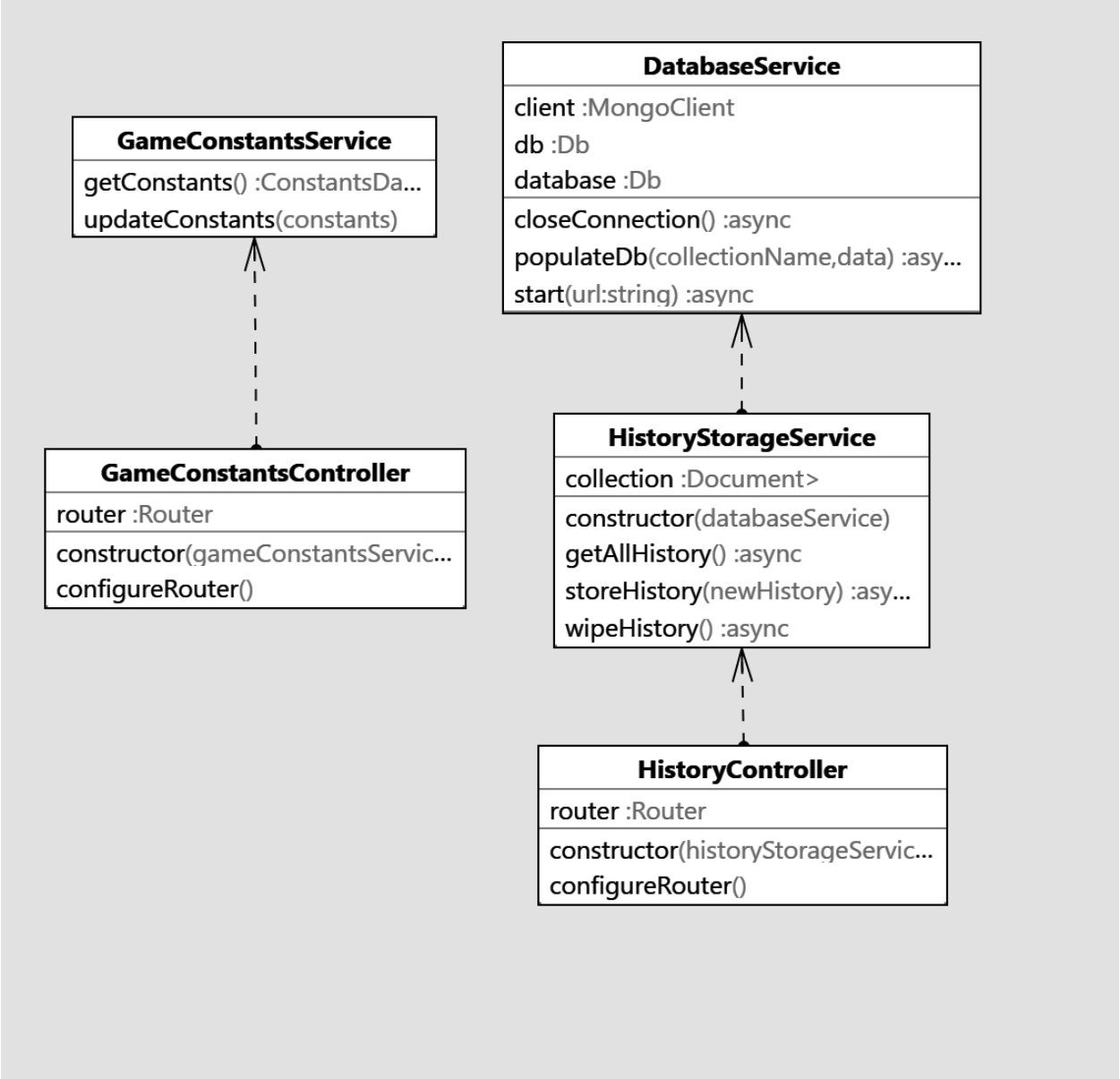
Serveur: Persistance des données de jeu
GameStorageService a la responsabilité de communiquer avec MongoDB pour gérer les jeux (suppression, ajout, modification, etc…). Pour cela, DatabaseService sert d'interface entre le serveur dynamique et MongoDB. On utilise ce service pour établir une connexion avec la base de données et FileSystemManager afin de lire des fichiers JSON. GameStorageService s'occupe aussi des images, qui sont sauvegardées localement sur le serveur dynamique.

Diagramme de classe - Persistance des données de jeu (serveur)



Serveur: Persistance des constantes et de l'historique
HistoryStorageService a la responsabilité de communiquer avec MongoDB pour gérer l'historique (suppression, ajout, modification, etc…). Il reprend également le service DatabaseService, dont le fonctionnement a été présenté dans la section de persistance des données. Les requêtes HTTP sont traitées par le contrôleur HistoryController. C'est le service GameConstantsService qui s'occupe de la gestion des constantes. Ses responsabilités consistent à lire et à écrire dans un fichier sauvegardé localement, comme les images ou l'identifiant du dernier jeu créé (fichier lastGameId.txt). Les requêtes HTTP sont traitées par le contrôleur GameConstantsController.

Diagramme de classe - Persistance des constantes et de l'historique (serveur)



5. Vue de déploiement

Le diagramme ci-dessous illustre la configuration finale de déploiement concret de l'ensemble du système.

Diagramme de déploiement