Document d'architecture logicielle

Version 1.2

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

Table des matières

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

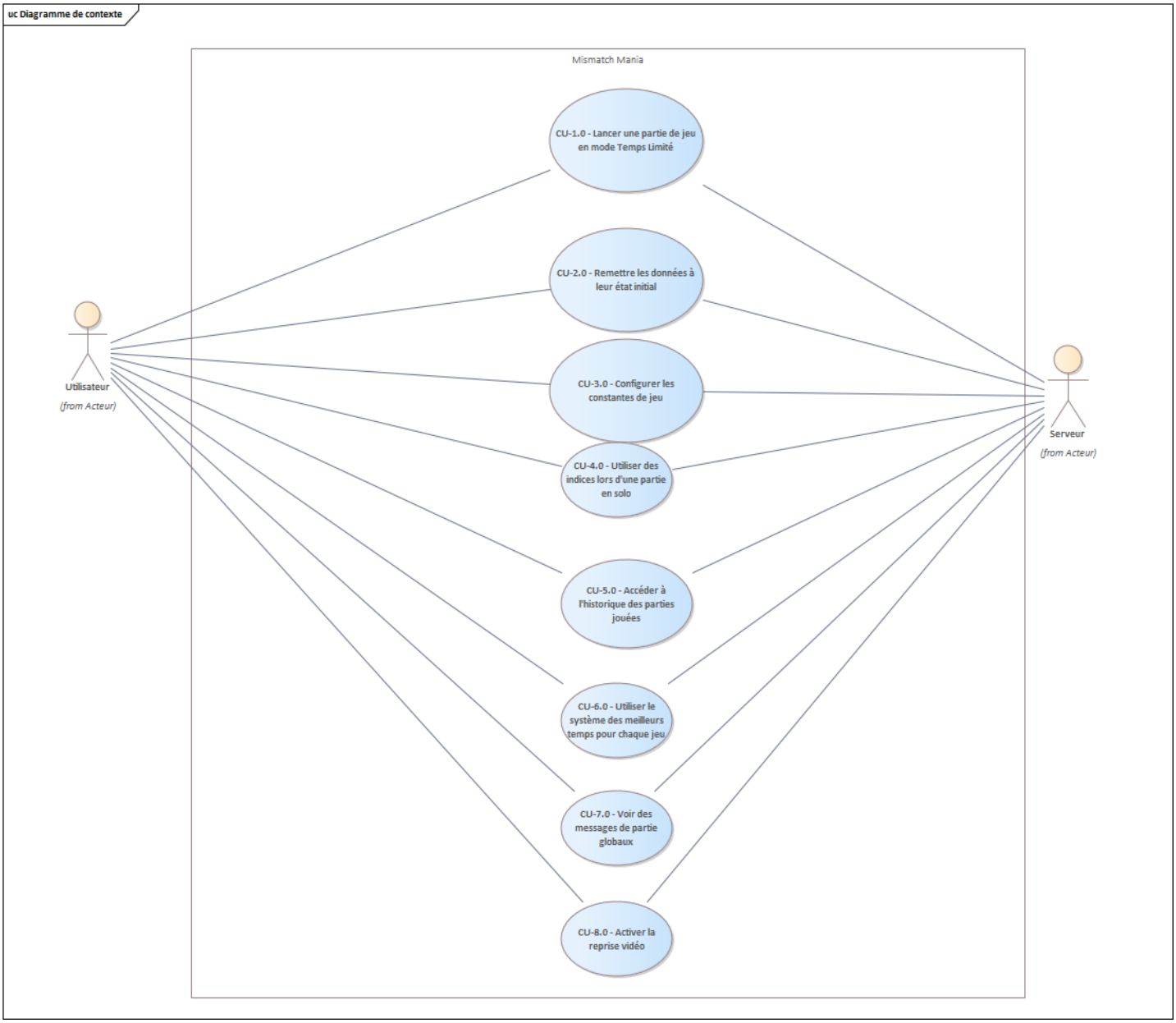


Diagramme de contexte CU1.0 - Lancer une partie de jeu en mode Temps Limité

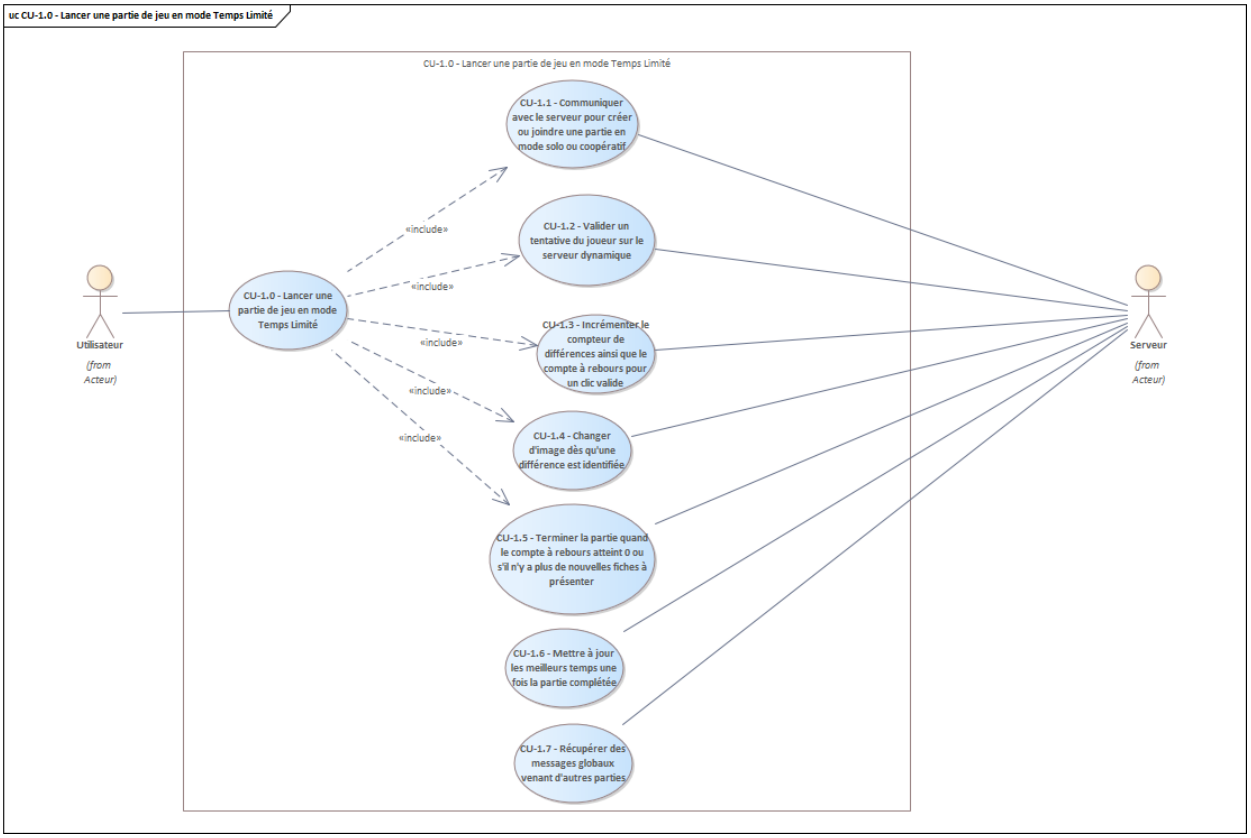


Diagramme de contexte CU2.0 - Remettre les données à leur état initial

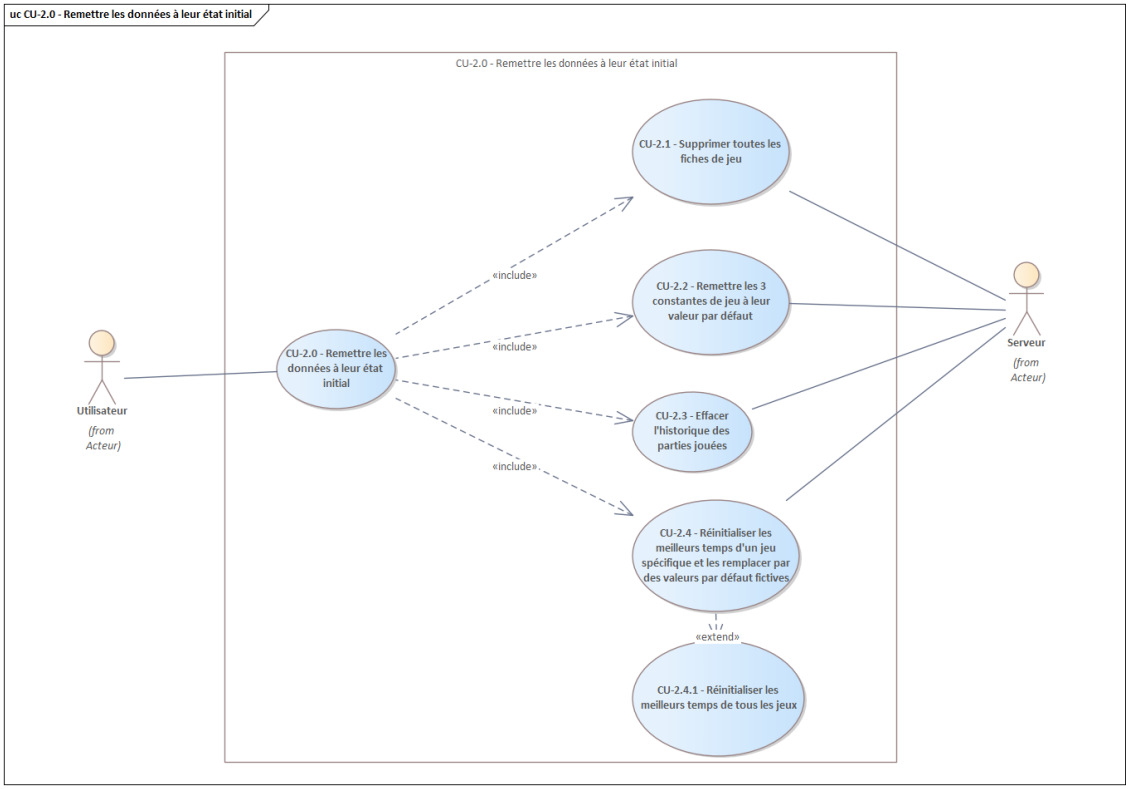


Diagramme de contexte CU3.0 - Configurer les constantes de jeu

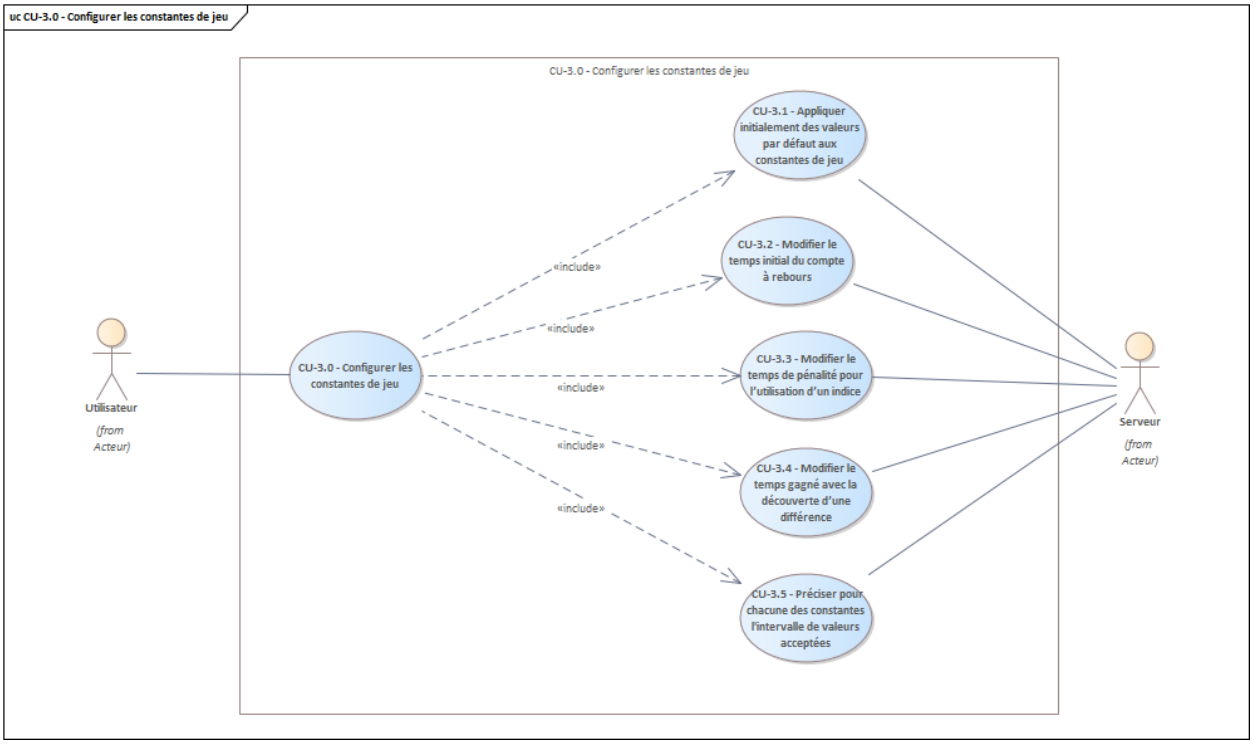


Diagramme de contexte CU4.0 - Utiliser des indices lors d'une partie en solo

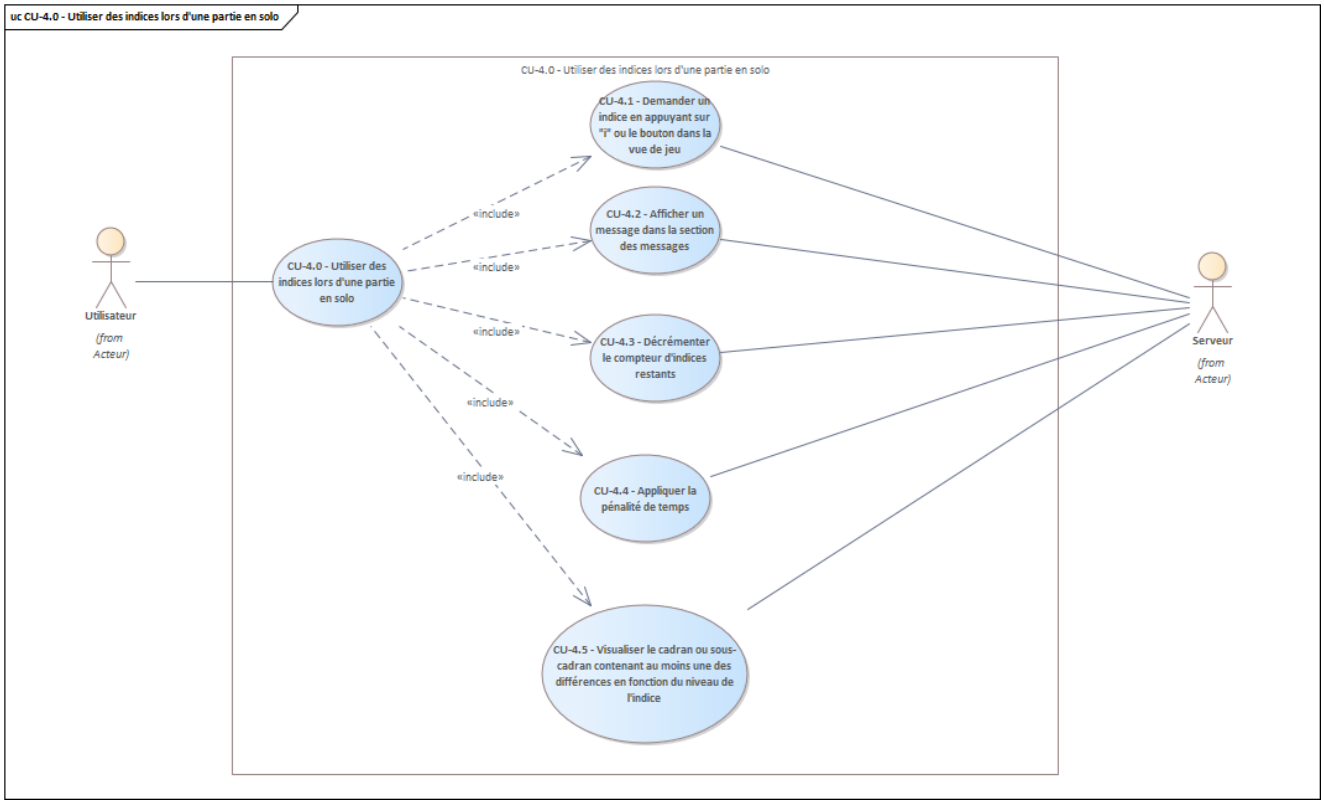


Diagramme de contexte CU5.0 - Accéder à l'historique des parties jouées

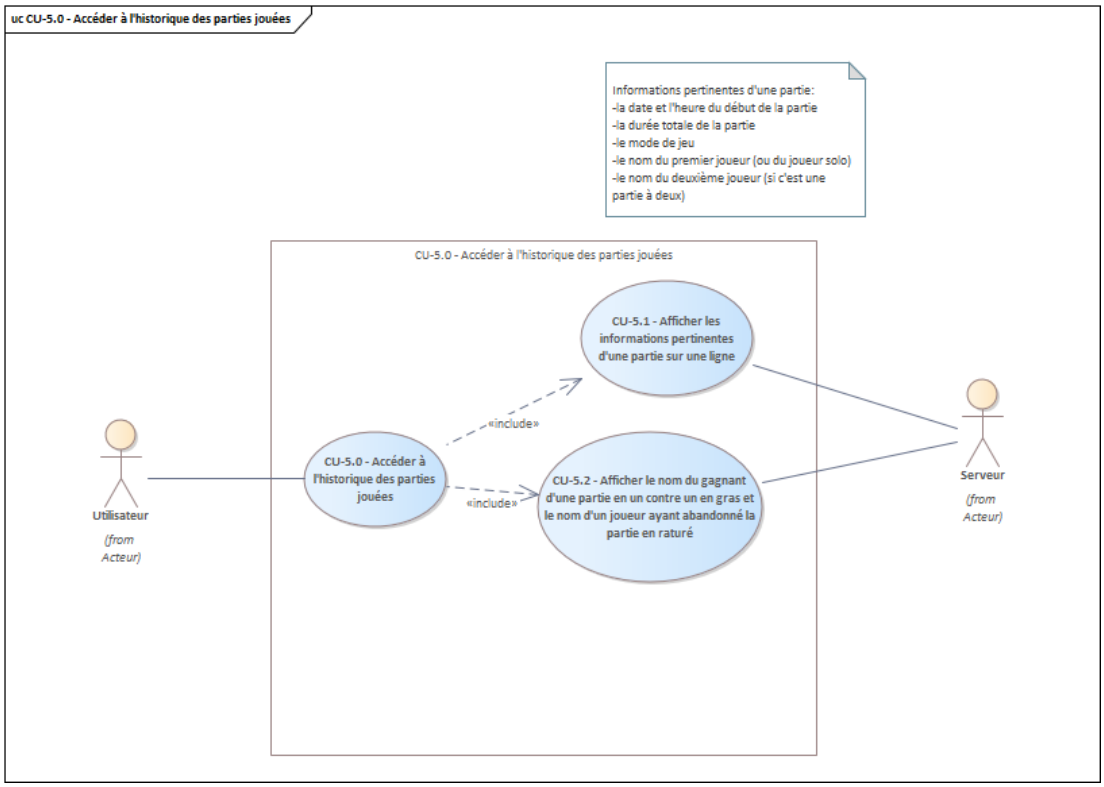


Diagramme de contexte CU6.0 - Utiliser le systeme des meilleurs temps pour chaque jeu

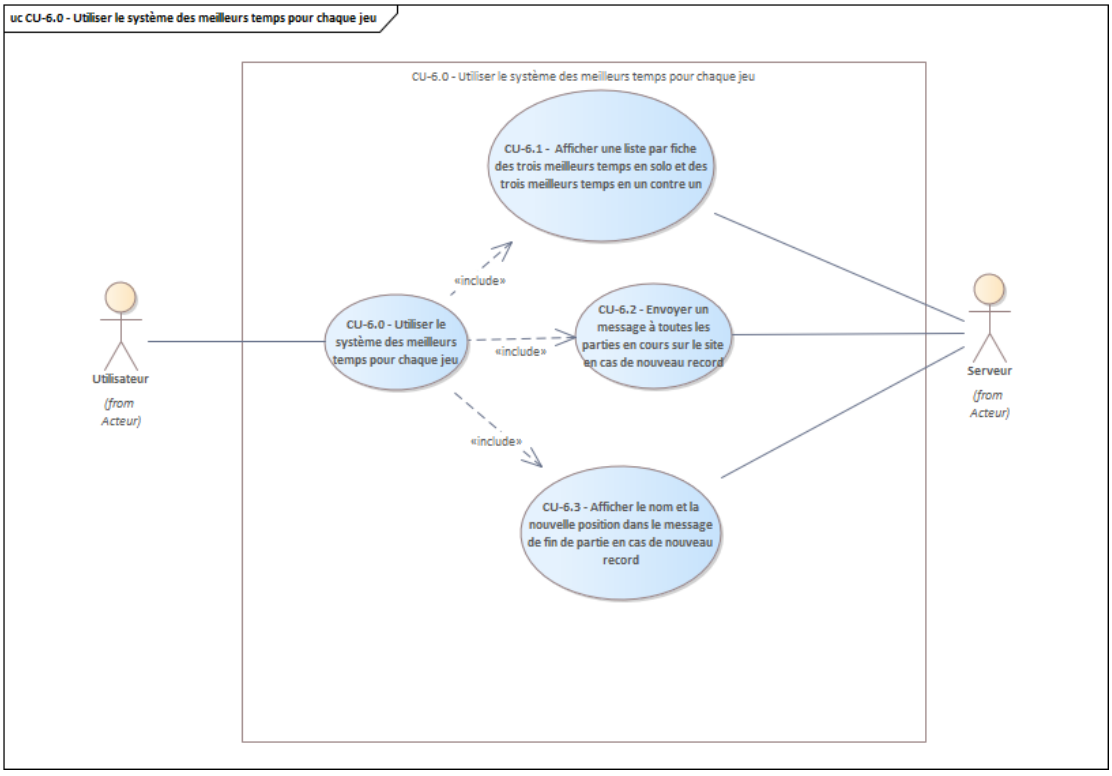


Diagramme de contexte CU7.0 - Voir des messages de partie globaux

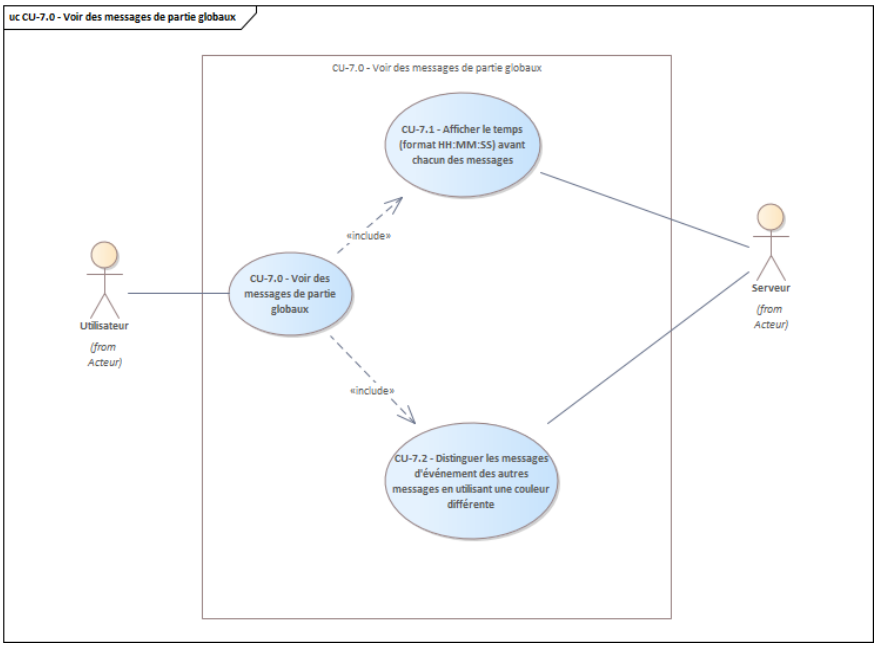
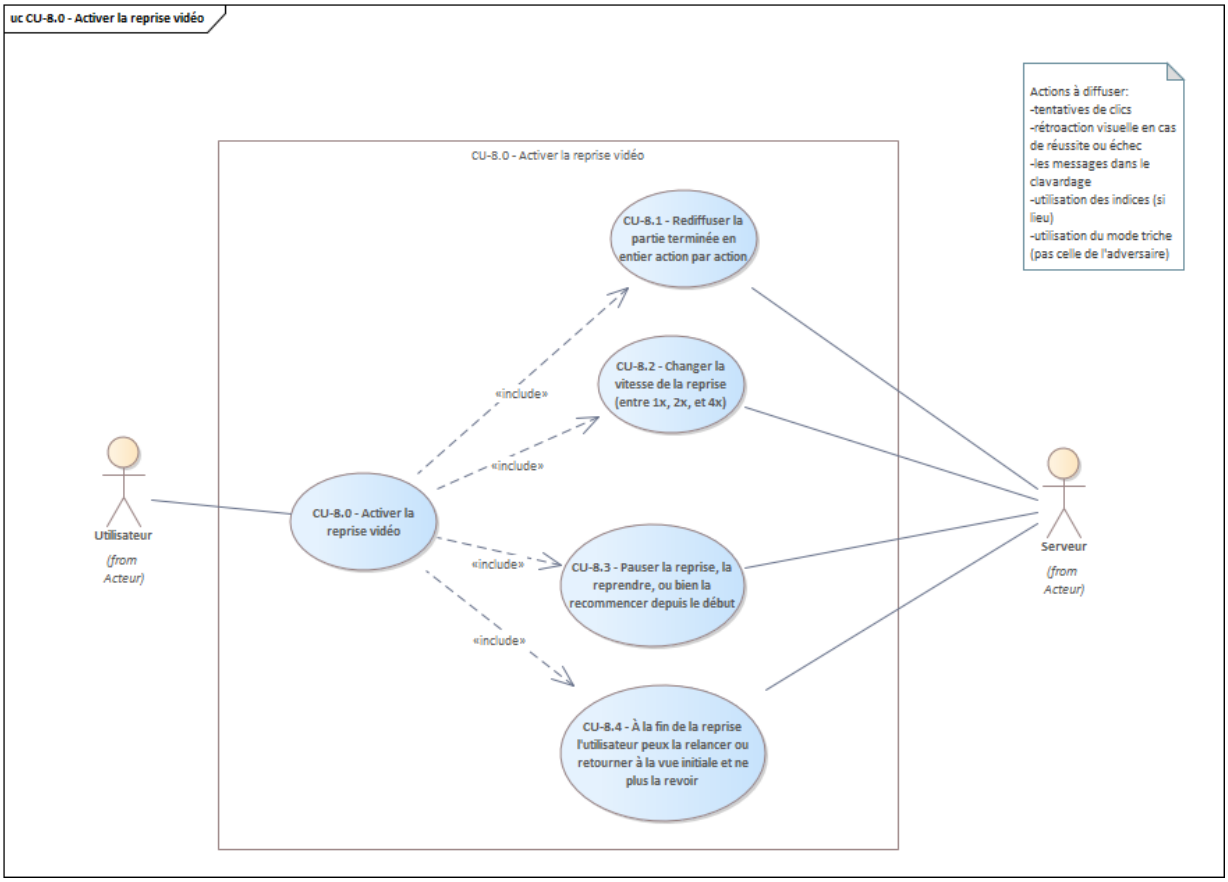


Diagramme de contexte CU8.0 - Activer la reprise vidéo



3. Vue des processus

Les diagrammes qui suivent insistent sur les processus principaux des fonctionnalités du Sprint 3 en illustrant leurs interactions dans un format séquentiel. Les diagrammes regroupent chacun des fonctionnalités selon leur contexte d'utilisation.

Le premier reflète le déroulement d'une partie en mode Temps Limité et précise également l'utilisation des indices en solo. On crée initalement une *room* avec Socket.io afin d'envoyer des informations entre le client et le serveur de façon bidirectionnelle. C'est ainsi que s'effectuent les interactions précisées dans le diagramme, telles que valider une différence, ou bien activer l'utilisation d'un indice. On précise également le comportement du processus une fois avoir complété une partie.On vérifie s'il faut mettre à jour les meilleurs temps pour la fiche concernée, et on sauvegarde le nouveau record le cas échéant. Enfin, le serveur enregistre l'historique du jeu.

Le deuxième diagramme est plus succinct et illustre les nouvelles fonctionnalités concernant la page de configuration, c'est-à-dire modifier les constantes de jeu et réinitialiser les meilleurs temps. Tel que demandé, les meilleurs temps sont sauvegardés sur MongoDB, ainsi que l'historique des parties.

Le troisième diagramme se concentre sur le Mode Reprise. Avant de lancer un jeu, on précise également qu'il est désormais possible pour l'utilisateur de consulter les meilleurs temps pour chacune des fiches. Le déroulement de la partie originale est omis afin d'améliorer la clarté du document. L'utilisateur lance le Mode Reprise, et les événements du jeu sont affichés une par une, en ordre chronologique.

Diagramme d’interaction - Mode Temps Limité et utilisation des indices

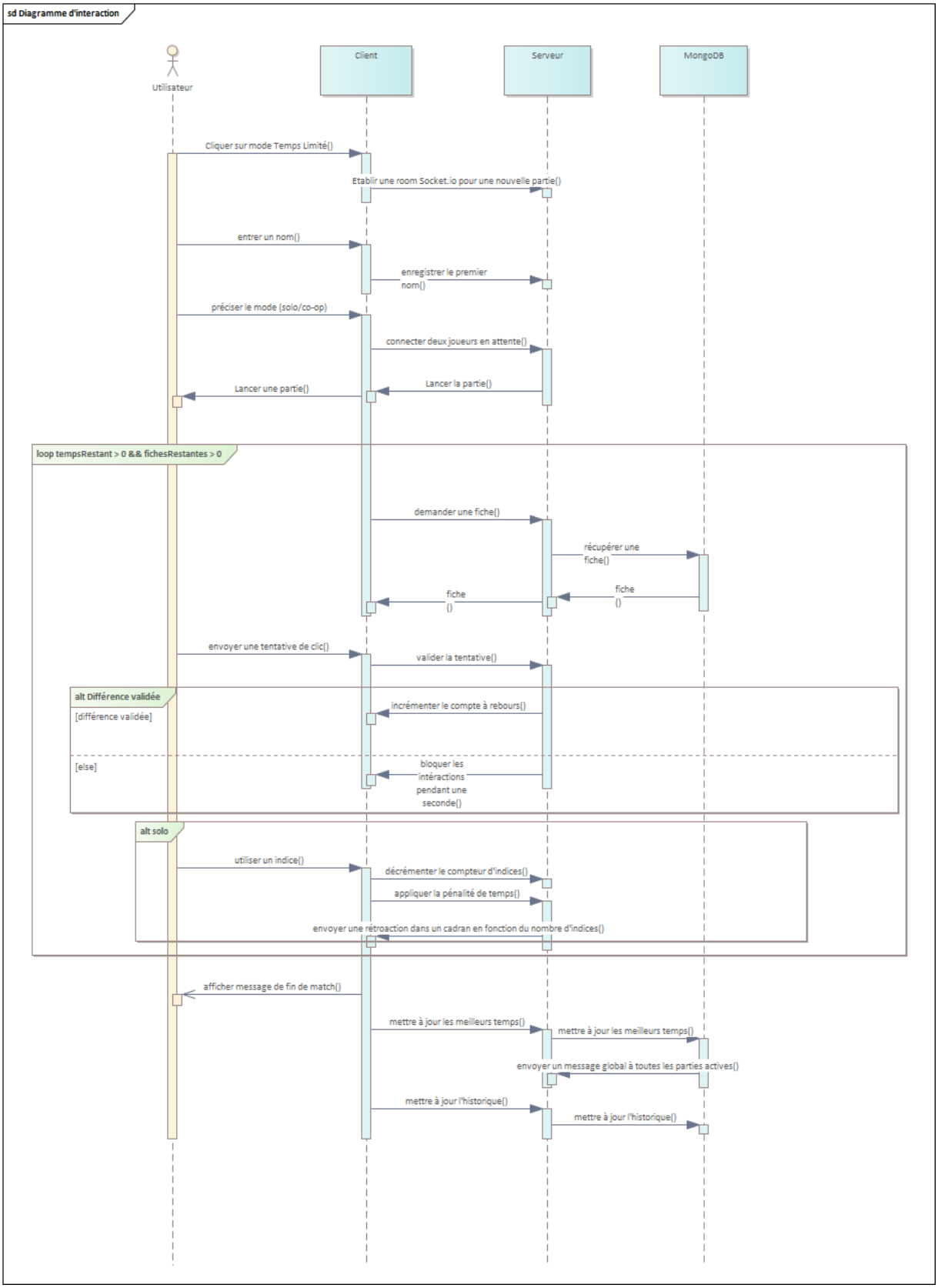


Diagramme d'interaction - Constantes de jeu et historique

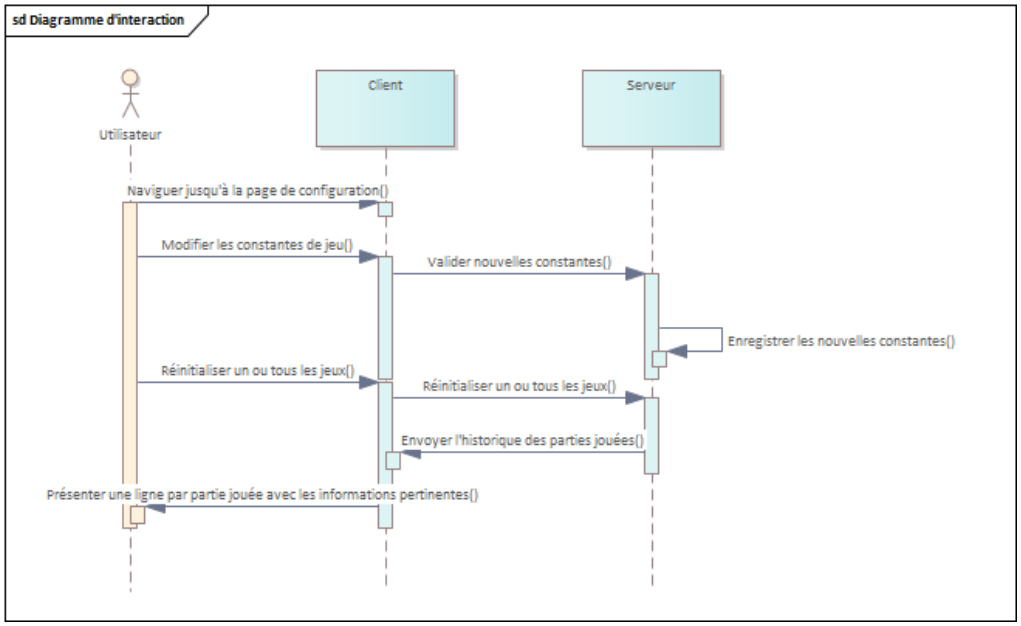
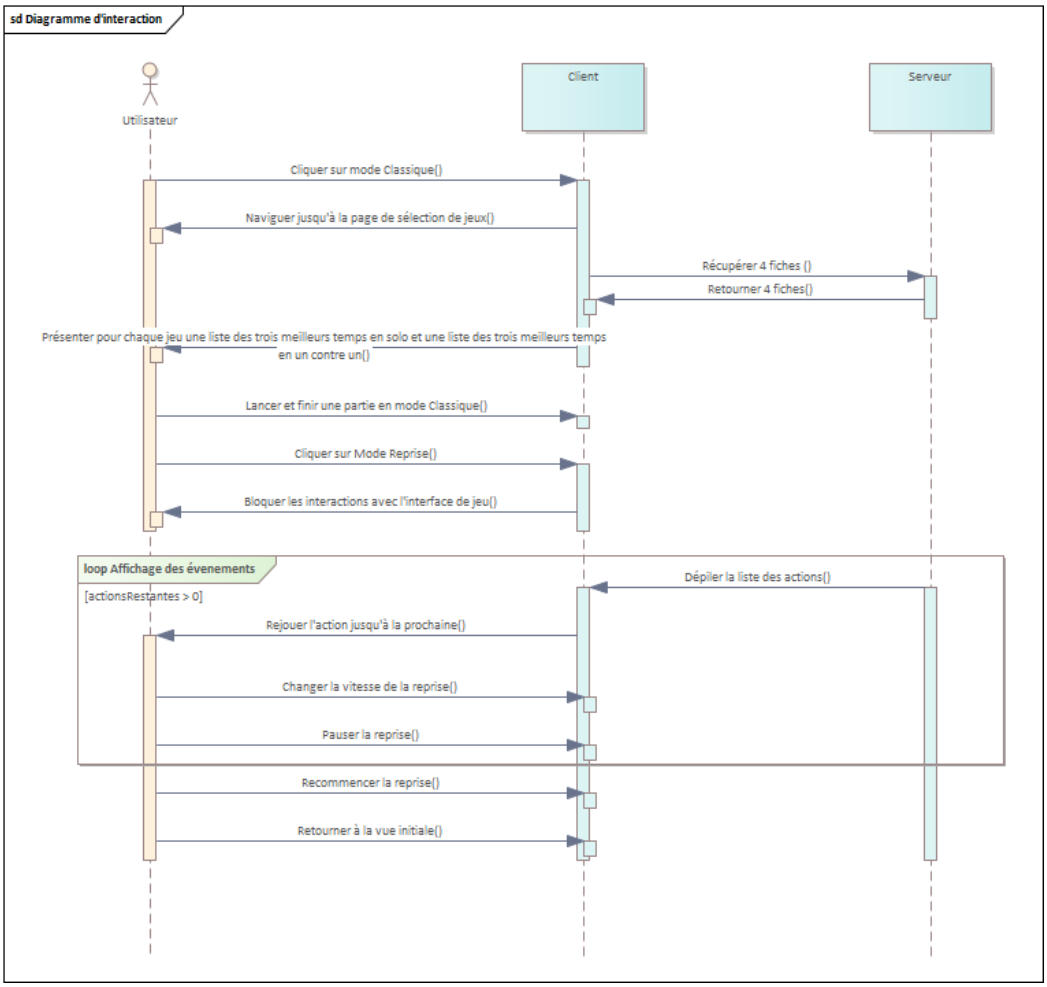


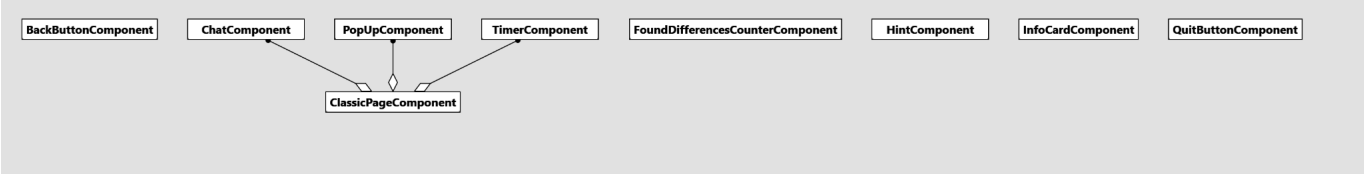
Diagramme d'interaction - Mode Reprise

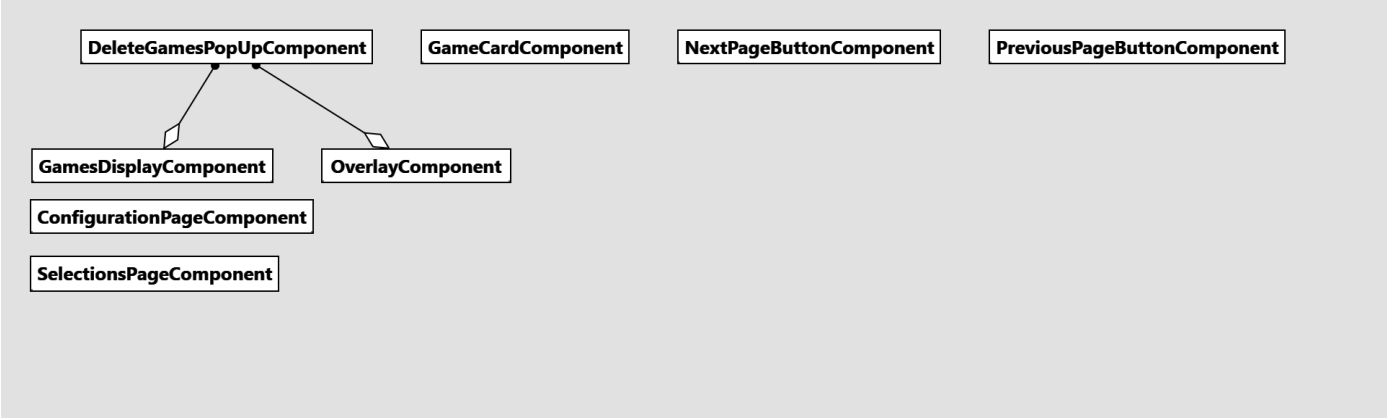


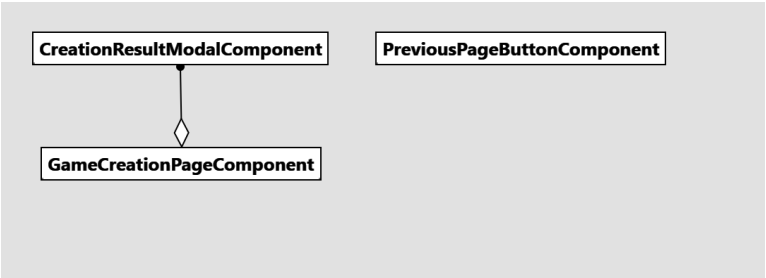
4. Vue logique

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 un pop-up afin de confirmer son choix.

Diagrammes de classe - Vue

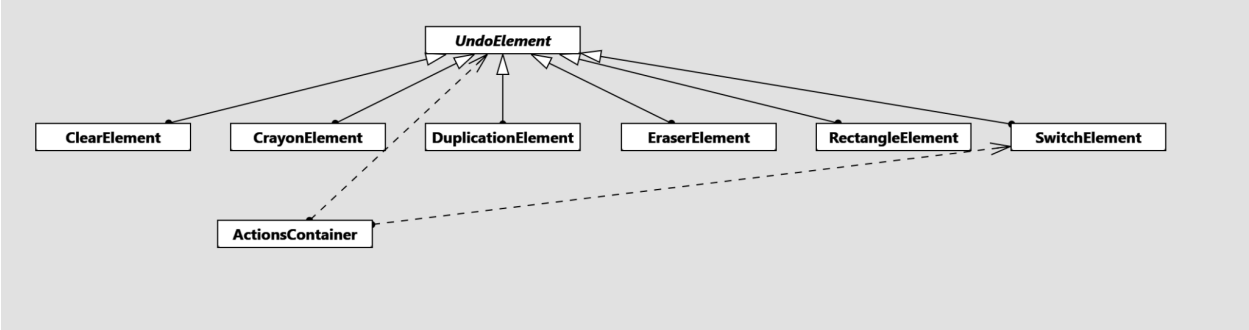


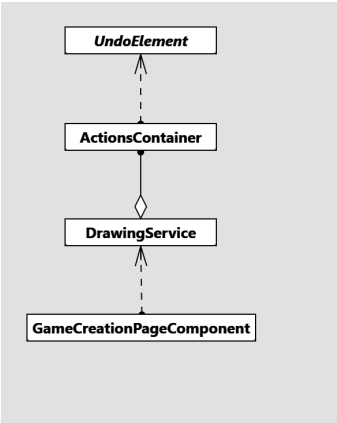




Client: Logique de modification de l'avant-plan
Le diagramme de classes ci-dessous illustre les classes pertinentes pour modifier l'avant-plan dans la page de création de jeu. L'utilisation du polymorphisme permet de regrouper toutes les actions effectuées dans un conteneur, et les refaire/défaire avec 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).

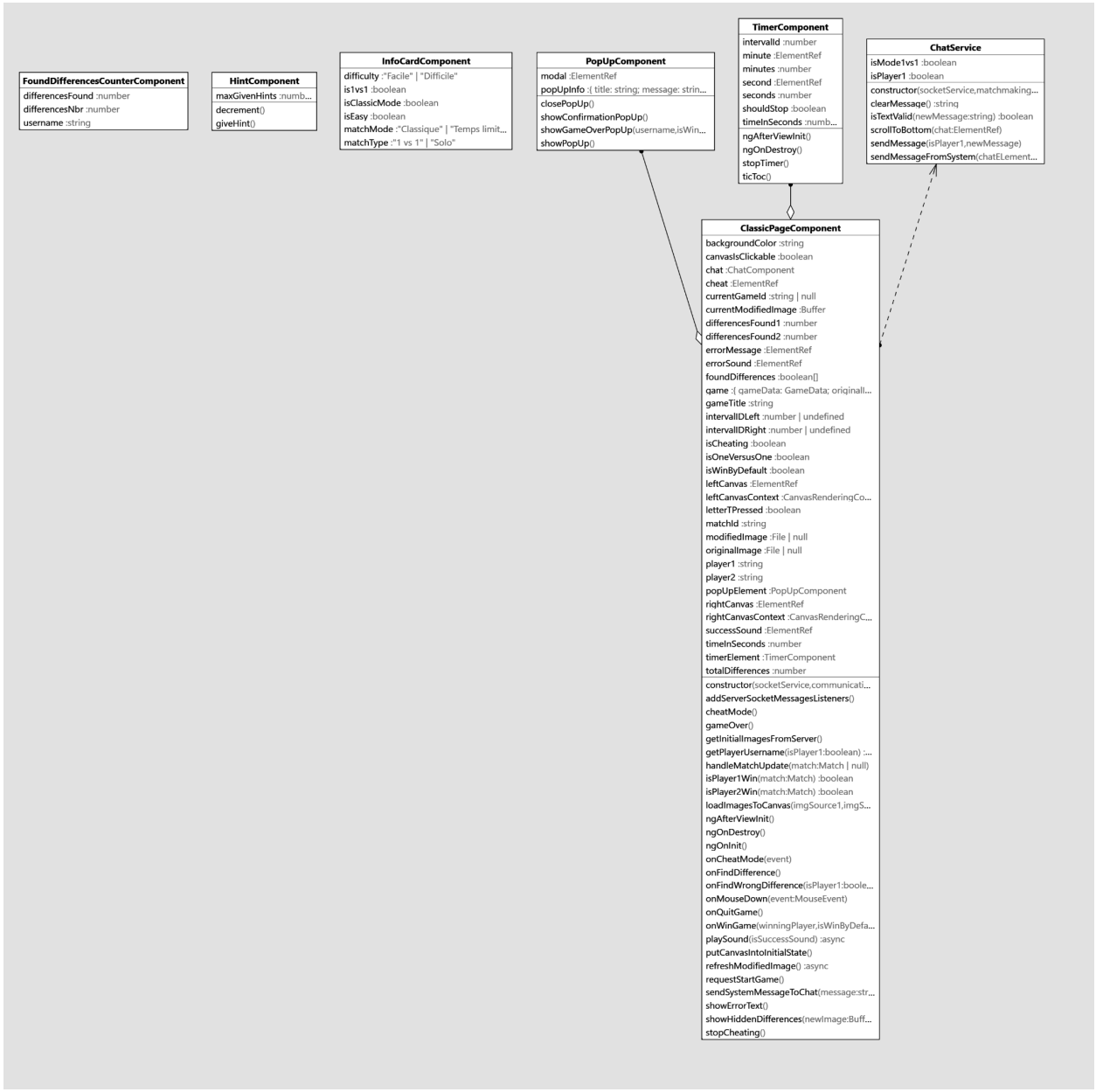
Diagrammes de classe - Logique de modification de l'avant-plan





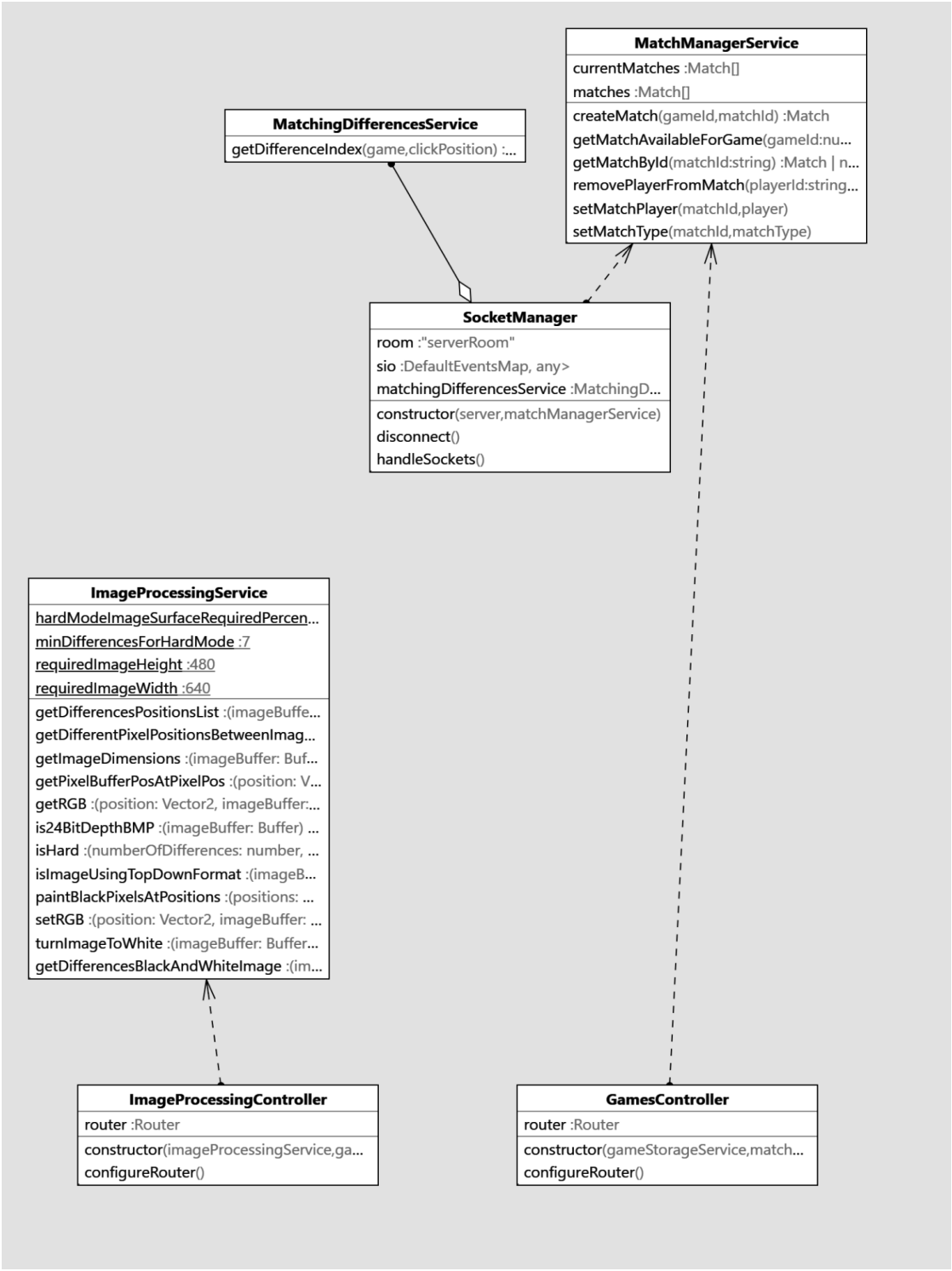
Client: Logique du fonctionnement d'une partie Classique
La page ClassicPageComponent est utilisée pour le mode Solo aussi bien que 1v1. Les deux figures qui suivent illustrent les classes du côté client et serveur pour implémenter le mode Classique (Solo et 1v1). Le service MatchmakingService envoie les requêtes Socket.io afin de mettre à jour les informations d'une partie en cours. Le service ImageManipulationService est également utilisé pour charger les images .bmp et faire clignoter les différences entre autres.

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



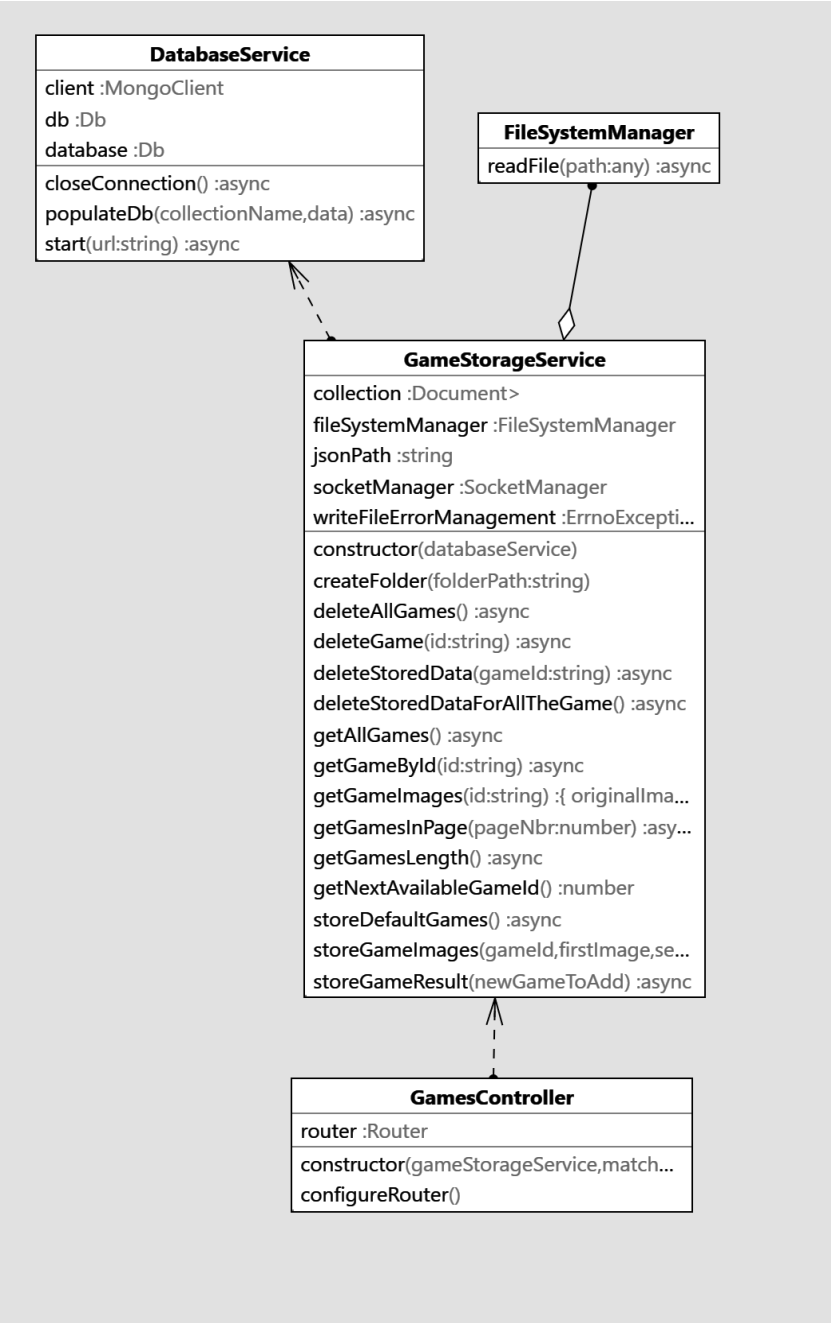
Serveur: Logique de fonctionnement d'une partie Classique
Du côté serveur, les services MatchManagerService ainsi que SocketManagerService s'occupent de la réception et l'envoi des requêtes Socket.io lors d'une partie. Pour calculer si une tentative de clic est valide, on utilise également le MatchingDifferencesService.

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



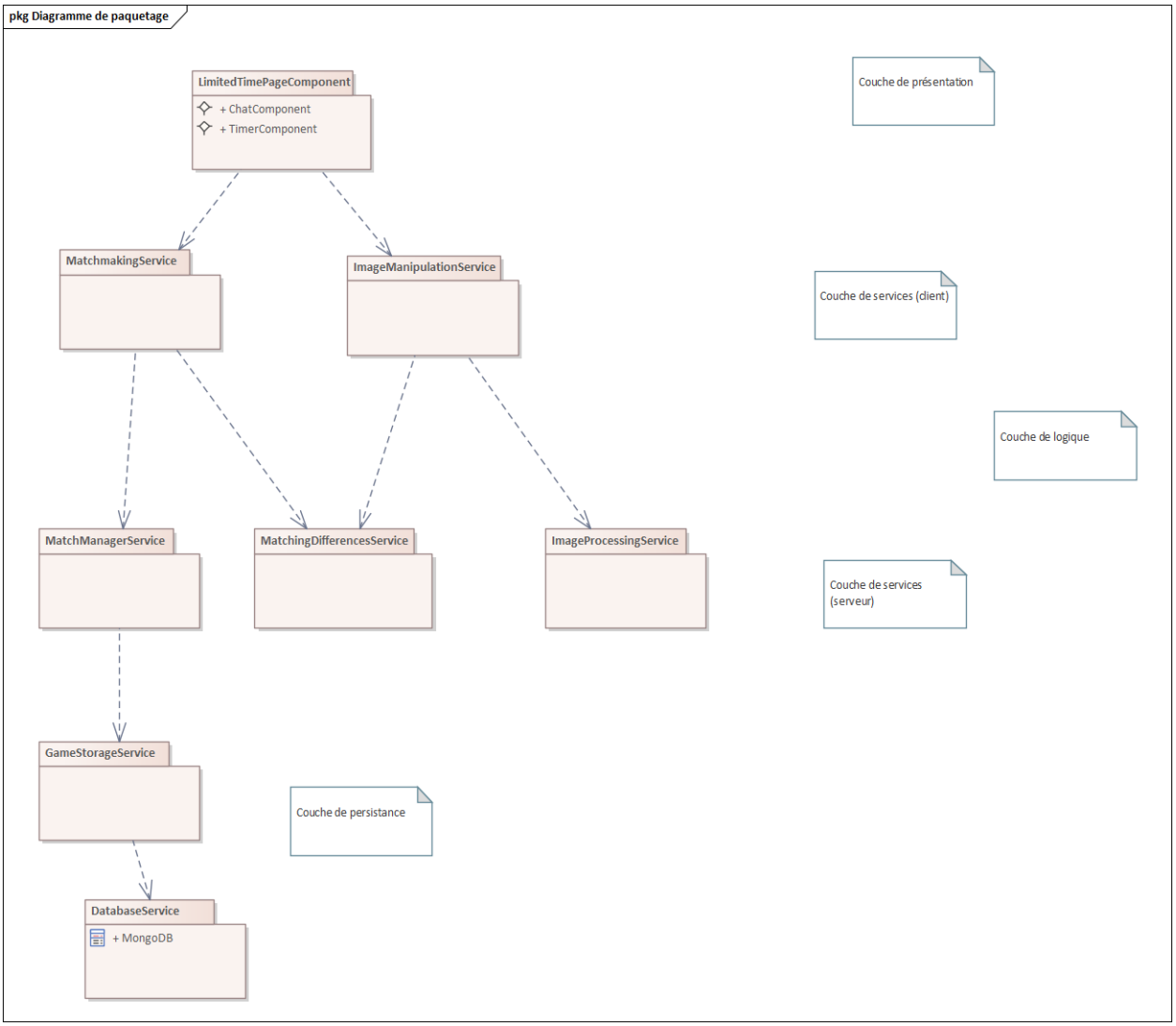
Serveur: Persistance des données (MongoDB)
GameStorageService a la responsabilité de communiquer avec MongoDB pour gérer les jeux (suppression, ajout, modification, etc…). Pour cela, DatabaseService sert d'interface entre le serveur dynamique et MongoDB. On utilise ce service pour établir une connexion 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 (MongoDB)



Fonctionnement du mode Temps Limité
Afin de ne pas surcharger la ClassicPageComponent, on a créé une nouvelle page appelée LimitedTimePageComponent afin d'implémenter le mode Temps Limité séparément. On remarque néanmoins que celle-ci utilise aussi le MatchmakingService ainsi qu'ImageManipulationService. Cette fois, il faut fournir plusieurs fiches au client, et configurer la logique des sockets afin de permettre aux deux joueurs de jouer simultanément. Dans le mode coopératif, il faut également traiter le cas ou l'un des deux joueurs quitte la partie en transformant la partie en mode solo.

Diagramme de paquetage - Fonctionnement du mode Temps Limité



Fonctionnement du Mode Reprise

La logique de contrôle du Mode Reprise est gérée par le service ReplayService, qui est instancié avec des références aux deux canvas sur la page de jeu. Tout au long du jeu, il est possible d'empiler des objets à la liste SavedActions, qui contient tous les événements effectués depuis le début du match.

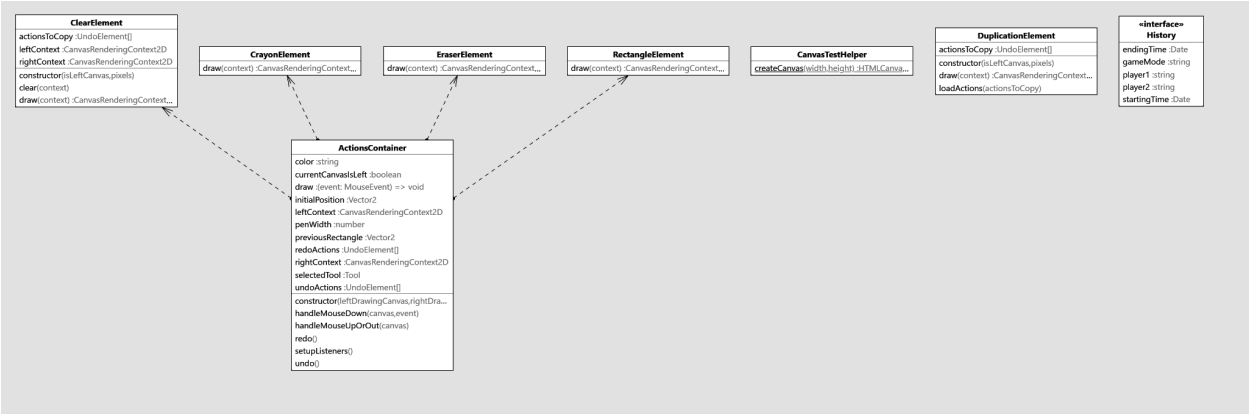
Afin de stocker plusieurs événements différents dans le même conteneur (ajout d'un message dans le clavardoir, détection d'une différence, utilisation du mode Triche, etc…), l'utilisation du polymorphisme et de l'héritage est donc nécessaire.

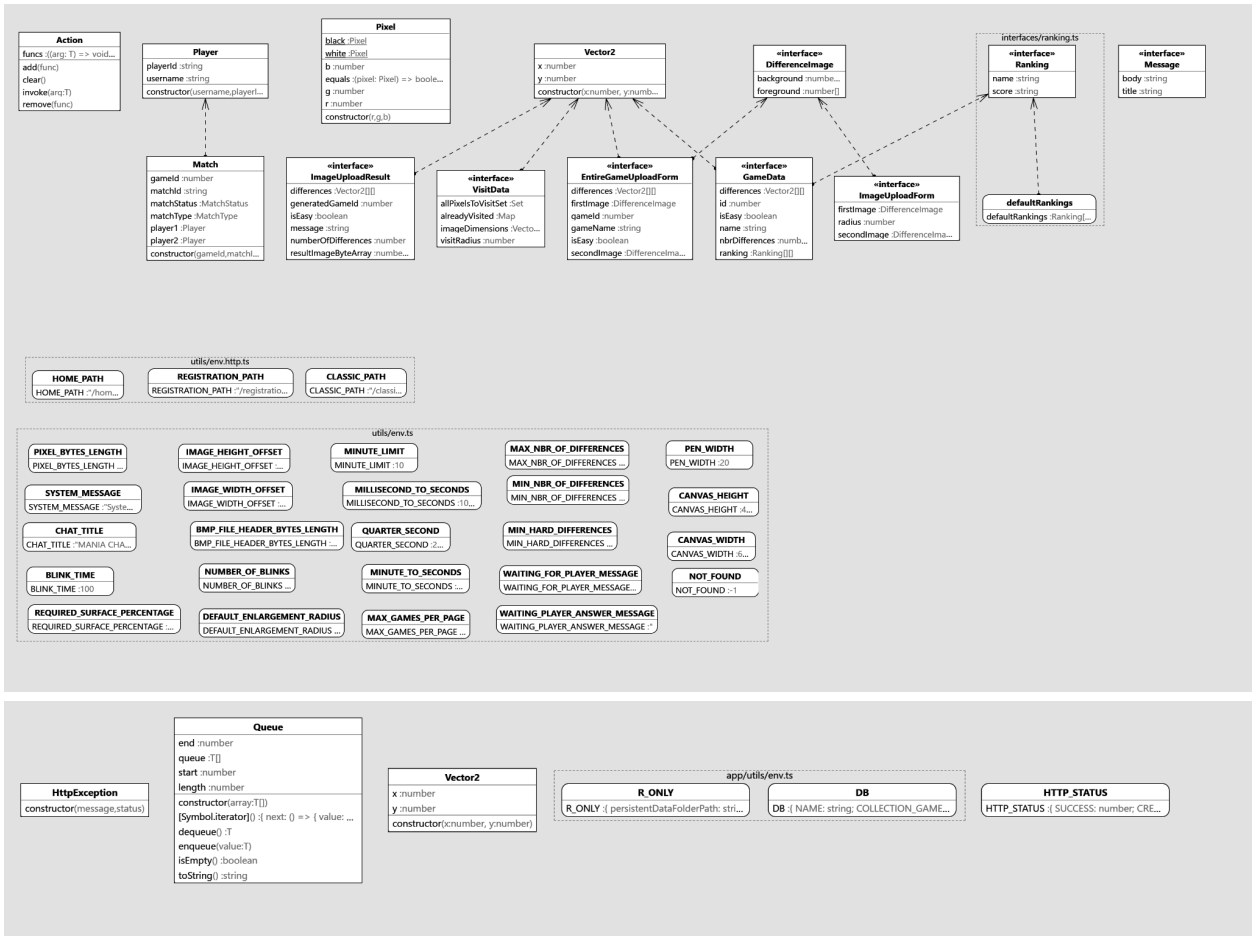
Chaque élément ajouté contient le temps où il a été effectué, ainsi qu'un enum indiquant le joueur qui doit rejouer l'action. (ex: un message global est visible aux deux joueurs, mais l'utilisation du Mode Triche doit seulement apparaître sur l'écran du joueur l'ayant activé.).

Constantes et classes auxiliaires

Le client et le serveur ont tous les deux leurs propres fichiers de constantes. Pour les constantes et classes partagées, on utilise le dossier 'Common' pour les rendre accessibles au client et au serveur.

Diagramme de classes - Constantes et classes auxiliaires (Client, Common, et Serveur respectivement)

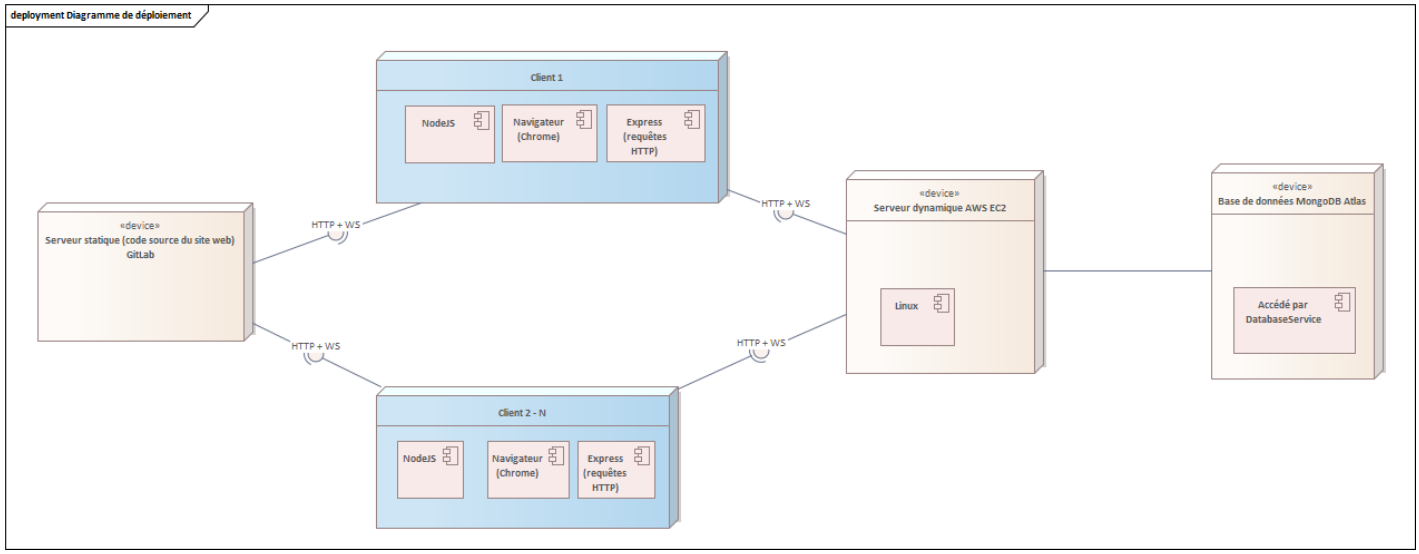




5. Vue de déploiement

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

Diagramme de déploiement


deployment Diagramme de déploiement
Client 1
NodeJS
Navigateur (Chrome)
Express (requêtes HTTP)
«device»
Serveur statique (code source du site web) GitLab
HTTP + WS
HTTP + WS
«device»
Serveur dynamique AWS EC2
Linux
«device»
Base de données MongoDB Atlas
Accédé par DatabaseService
HTTP + WS
HTTP + WS
Client 2 - N
NodeJS
Navigateur (Chrome)
Express (requêtes HTTP)