Conception Preliminaire

Table des matières

Division de la solution en différents composants

L'explication des interfaces fournies par chaque composant

La spécification des interfaces fournies par chaque composant

Le diagramme de séquence qui valident ces interfaces

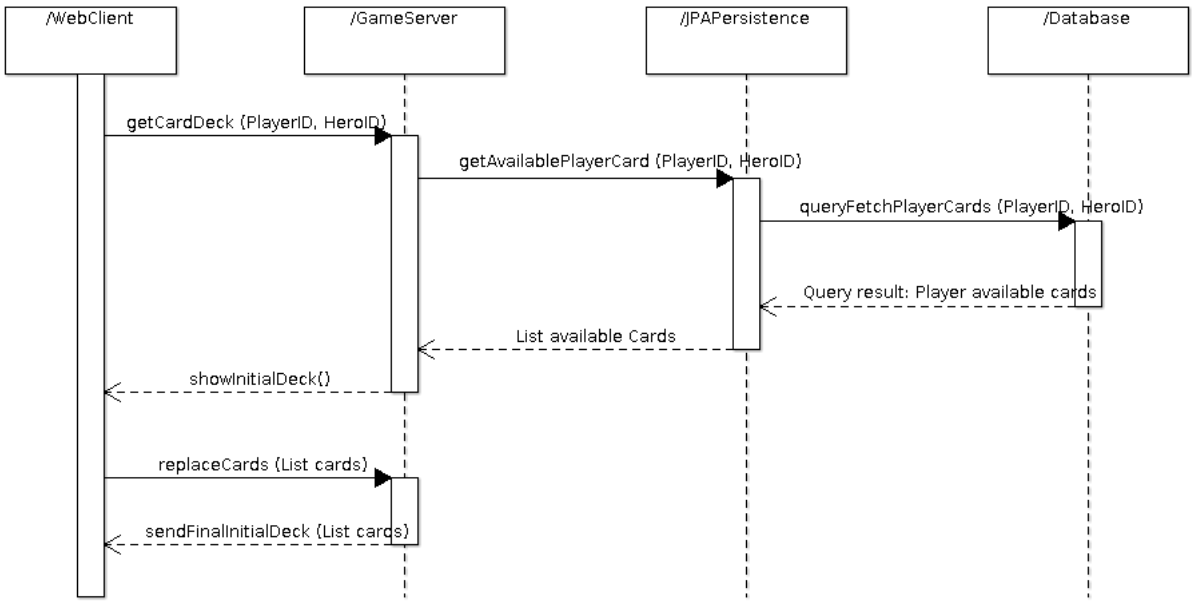
Nous avons fait un diagramme de séquence par scénario afin de distinguer les différentes cas d'intéractions possibles. Nous n'avons pas représenté les messages liés a feeback car nous avons considéré que cela est géré au niveau du WebClient. Par exemple : si un joueur essaie de choisir une carte mais il n'a pas assez de mana pour celle-ci, la première vérification se fait au niveau du WebClient pour notifier au joueur qu'il n'a pas assez de mana. Mais il vrai qu'il n'est pas très sécurisé d'effectuer les vérifications uniquement côté client car les clients peuvent tricher. Il est donc nécéssaire de faire une double vérification des préconditions côté serveur, mais nous ne les avions pas représenté dans les diagrammes de séquences afin de ne pas les alourdir.

Scénario : Récupération du premier 'Main de cartes'

Ce scénario illustre l'intéraction entre les 4 composants lorsque de le WebClient demande au serveur de générer son premier *'main de cartes'*.

Les cartes qui sont disponible pour le héros choisit, sont stockés dans la bases de données. Certaines cartes sont débloquées (cartes plus puissantes) en fonction du niveau qu'à le joueur avec le héros choisis. C'est pour cela que dans la fonction *getCardDeck (PlayerID, HeroID)* on passe en parametre l'identifiant du joueur et du héros.

Une fois la requête exécuté, JPA renvoie au GameServer les différents cartes disponibles pour ce héros sous forme de *liste*. Le serveur fait générer ensuite la main de cartes aléatoire à partir de *'List availableCards'* et le renvoie au WebClient.



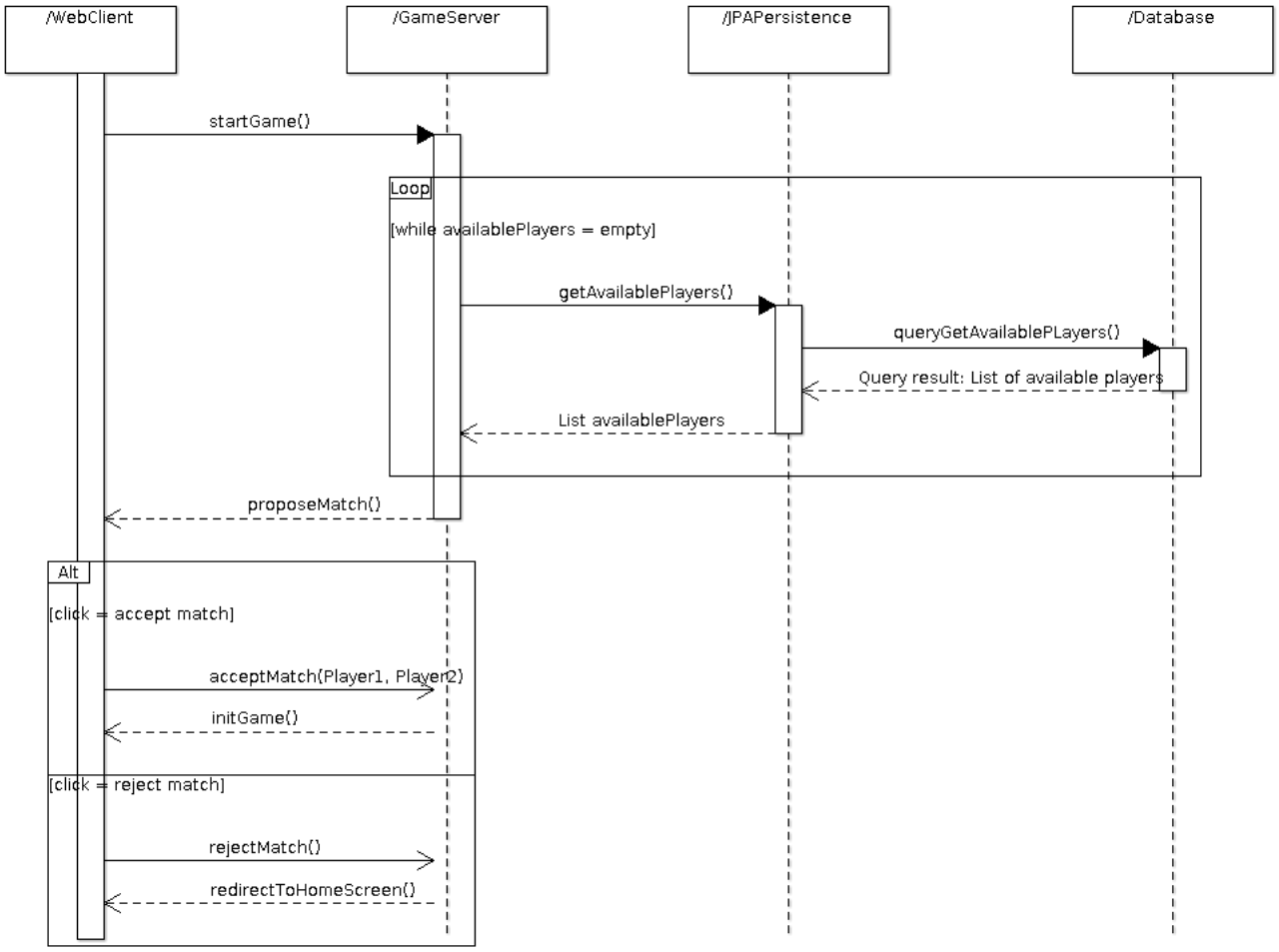
scenario 1

Scénario : Demander le démarrage d'une partie et matchmaking aléatoire

Ce scénario illustre l'intéraction lorsqu'un joueur demande à entrer dans une partie. Lorsqu'un joueur demande à entrer dans une partie, le WebClient doit demander au serveur de récupérer la liste des joueurs en lignes **qui ne sont pas dans une partie** : *getAvailablePlayers()* de la base de données via JPAPersistence.

JPAPersistence récupère le résultat de la requête et renvoie l'envoie sous forme de liste au GameServeur : *List availablePlayers*. Le serveur tire un des joueurs de la liste aléatoirement et propose le match au WebClient.

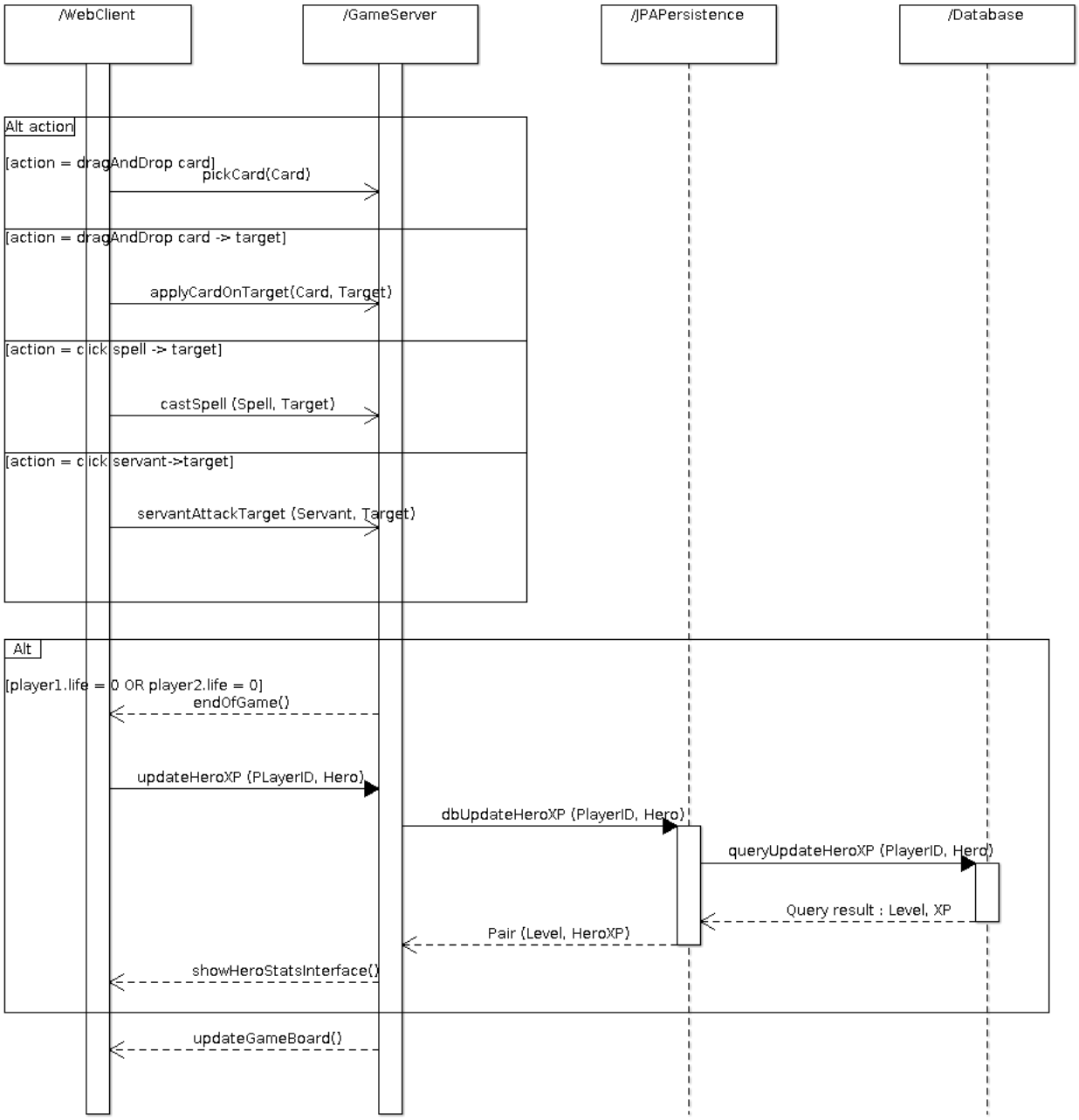
Le WebClient peut soit accepter ou rejecter me match.



scenario 2

Scénario : Différentes actions possibles lors d'un tour

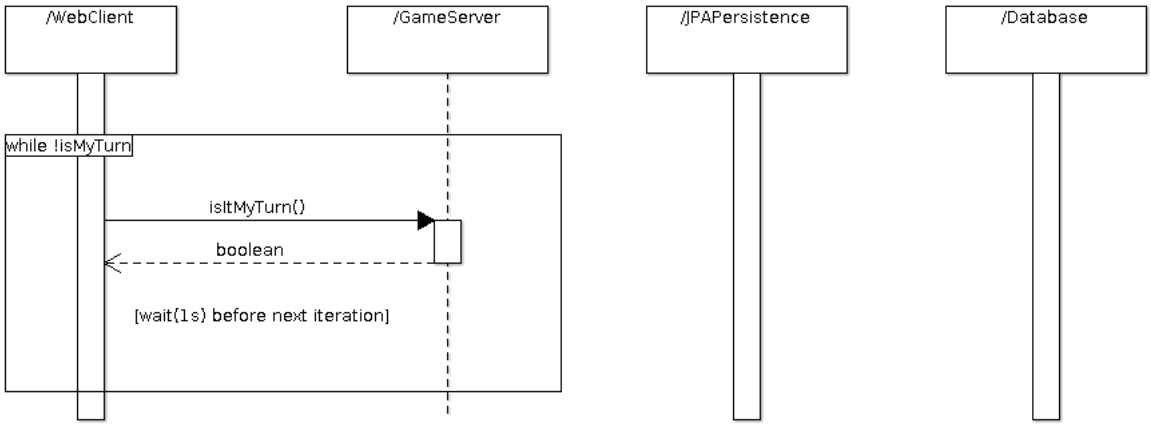
Ce scénario illustre les différentes actions possible que peut effectuer un joueur et les différentes données transmises au serveur en fonction de ces choix. Si les points de vie d'un des joueur vaut 0 alors c'est la fin de la partie et le serveur gère la demande de sauvegarde des points d'expériences acquis lors de la partie en bases de données. Lorsque ces données ont été correctemet sauvegardé en bases de données, le serveur met à jour l'interface du WebClient afin de montrer au joueur les statistique de son héros.



scenario 3

Scénario : Attente de mon tour

Dans ce scénario, on considère qu'on est le joueur1 et que le joueur2 est en train de jouer. Le WebClient contient une variable booleénne **isMyTurn**. Tant que celle-ci vaut *false* le WebClient demande continuellement au serveur si c'est son tour. Tant que le joueur1 est en attente, le WebClient est obligé de demander continuellement si c'est son tour, car le WebClient connaît le GameServeur mais le GameServer ne connaît pas de client. ON a donc une association unidirectionel du WebClient vers le GameServer. Cela veut dire que le GameServer ne peux pas envoyer de message au WebClient, il peut seulement lui répondre. Donc dès que le joueur2 termine son tour, le GameServer est incapable de donner lui-même la main au joueur1. D'où la nécéssité du WebClient du joueur1 de devoir demander en continue si c'est son tour.



scenario 4