Specification Algo/Systeme

Omer , Lounas

24 février 2025

1 Introduction

1.1 Objectif

L'objectif de ce document est de specifier les fonctionnalites, les exigences et le fonctionnement du jeu **6 qui prend !**, en detaillant ses règles, son architecture logicielle et son mode de fonctionnement. Ce document sert de reference pour les developpeurs, testeurs et contributeurs du projet.

1.2 Portee

Ce projet consiste en le developpement d'un jeu de cartes **6 qui prend !**, jouable en solo contre des bots ou en multijoueur en ligne. L'application vise a etre accessible via un navigateur web et potentiellement sur mobile a travers un developpement en **Godot (GDScript)**.

Le jeu suivra les règles classiques avec quelques ameliorations :

— Integration d'un mode **IA (bot)**
— Ajout d'un mode **multijoueur en ligne** avec gestion de salon.
— Implementation d'une **interface utilisateur interactive**

2 Description Generale

2.1 Perspective du Jeu

Le jeu sera base sur un modèle **client-serveur**, permettant de jouer a plusieurs joueurs en ligne ou contre une **IA (bot)**. Il comportera :

— Une interface interactive permettant aux joueurs de poser leurs cartes.
— Une gestion automatique des règles (placement des cartes, recuperation de rangees, comptage des tetes de clown).
— Un système de **salons** pour rejoindre ou creer des parties.

2.2 Interfaces Utilisateurs..... devrait etre cmpleter par les front

L'interface principale inclura :

— ecran d'accueil avec menu principal.
— Zone de jeu avec les rangees de cartes et la main du joueur.
— Affichage des scores et minuteur en mode multijoueur.
— ecran de fin de partie affichant les resultats.

3 Fonctionnalites

3.1 Fonctionnalites Joueurs

— Jouer contre d'autres joueurs (**mode multijoueur**).
— Jouer contre une **IA (bot)**.
— Rejoindre un **salon prive** ou une **partie rapide** en ligne.
— Selectionner une carte et la poser selon les règles du jeu.
— Afficher les scores et le classement en fin de partie.

3.2 Fonctionnalites Serveur

— Gestion des salons et sessions de jeu.
— Verification et validation des mouvements des joueurs.
— Mise a jour des scores et des rangees en temps reel.
— Gestion de l'IA et de son comportement dans les parties solo.
— Gestion des deconnexions et reconnexions des joueurs en cours de partie.

4 Exigences Fonctionnelles

4.1 Mecanique de Jeu

— Distribution des cartes : Chaque joueur reçoit **10 cartes aleatoires**.
— Mise en place des rangees : **4 cartes initiales** sont placees en ordre croissant.
— Tour de jeu :
  — Chaque joueur choisit une carte (avec un timer de 30sec) et la pose **face cachee**.
  — Les cartes sont revelees, triees et placees selon les règles.
— Si une carte devient la **6ème** dans une rangee, le joueur **prend les 5 cartes precedentes**.
— Si une carte est trop petite pour etre placee, le joueur **choisit une rangee** et la recupère.

4.2 Gestion des Scores

Chaque carte a un nombre de **tetes de clown** determinant la penalite du joueur. **Le but est d'avoir le moins de tetes de clown possible**.

4.3 Exceptions

Lors de l'implémentation du jeu, plusieurs cas exceptionnels doivent être gérés afin d'assurer une expérience de jeu fluide et sans interruption.

— **Expiration du temps de jeu** :
  — Si le temps imparti pour choisir une carte est écoulé et que le joueur n'a pas fait de choix, alors une carte est sélectionnée aléatoirement depuis sa main.
— **Déconnexion d'un joueur** :
  — Si un joueur se déconnecte en cours de partie, il est remplacé par un **bot** qui reprend le même score et continue la partie.
— **Égalité en fin de partie** :
  — Si plusieurs joueurs ont le même nombre de têtes de clown à la fin de la partie, alors le joueur ayant accumulé le plus petit score sur la dernière manche est déclaré vainqueur.

  - Si l'égalité persiste, la priorité revient au joueur ayant le moins de têtes de clown sur les 3 dernières manches.
- **Problème de placement d'une carte** :
  - Si une carte ne peut être placée sur aucune rangée (c'est-à-dire qu'elle est inférieure à toutes les dernières cartes des rangées) et que le joueur ne choisit pas manuellement dans un délai imparti, une rangée aleatoirement lui sera attribuée.
- **Joueur inactif** :
  - Si un joueur est inactif pendant plusieurs manches consécutives (exemple : 3 tours), il est automatiquement exclu et remplacé par un bot.

5 Conception Logicielle et Structure des Classes

Les classes principales du projet incluent :
- **Carte** : valeur, nombre de tetes.
- **Deck** : gestion des 104 cartes, melange et distribution.
- **Rang** : gestion d'une rangee sur la table.
- **Table** : gestion des 4 rangees de jeu.
- **Main** : cartes detenues par un joueur.
- **Joueur** : ID, main, score, nom, nombre d'AFK.
- **Game6Takes** : gestion du deroulement du jeu.

Les classes principales du projet sont detaillees ci-dessous avec leurs attributs et methodes associees :

5.1 Classe Carte

- **Attributs** :
  - `valeur` : entier representant la valeur de la carte.
  - `nbTetes` : entier indiquant le nombre de têtes de clown associees.
- **Methodes** :
  - `calculerTetes()` : Determine le nombre de têtes de clown en fonction des règles du jeu.

5.2 Classe Deck

- **Attributs** :
  - `cartes[]` : liste contenant toutes les cartes du jeu.
- **Methodes** :
  - `creerCarte()` : Genère les 104 cartes du jeu.
  - `melanger()` : Melange aleatoirement les cartes du deck.
  - `distribuer()` : Donne un ensemble de cartes aux joueurs.

5.3 Classe Rang

- **Attributs** :
  - `cartes[]` : liste de cartes appartenant a cette rangee.
- **Methodes** :
  - `ajouterCarte()` : Ajoute une carte a la rangee et verifie si elle atteint la 6ème position.

5.4 Classe Table

— **Attributs** :
  — `rangees[]` : tableau contenant les 4 rangees en cours de jeu.
— **Methodes** :
  — `trouverRangeeAdequate()` : Trouve la meilleure rangee pour placer une carte selon les règles du jeu.

5.5 Classe Main

— **Attributs** :
  — `cartes[]` : liste des cartes detenues par un joueur.
— **Methodes** :
  — `jouerCarte()` : Permet au joueur de poser une carte sur la table.

5.6 Classe Joueur

— **Attributs** :
  — `id` : identifiant unique du joueur.
  — `main` : ensemble de cartes que le joueur possède.
  — `score` : points de penalite accumules par le joueur.
  — `nom` : nom du joueur.
  — `nbAfk` : nombre de tours où le joueur a ete inactif.
— **Methodes** :
  — `updateScore()` : Met a jour le score du joueur.
  — `updateAfk()` : Incremente le compteur de tours inactifs.
  — `resetAfk()` : Reinitialise le compteur d'AFK.
  — `resetScore()` : Reinitialise le score du joueur.
  — `getMain()` : Retourne les cartes en main du joueur.
  — `getScoreJoueur()` : Retourne le score du joueur.
  — `getNom()` : Retourne le nom du joueur.

5.7 Classe Game6Takes

— **Attributs** :
  — `deck` : contient l'ensemble des cartes utilisees dans la partie.
  — `table` : instance representant la table de jeu.
  — `joueurs[]` : liste des joueurs participant a la partie.
  — `nbManche` : nombre total de manches dans la partie.
  — `nbTetes` : nombre total de têtes de clown collectees.
  — `mancheActuelle` : numero de la manche en cours.
— **Methodes** :
  — `jouerCarte()` : Gère le processus de jeu pour chaque joueur.
  — `checkCarte()` : Verifie la validite d'une carte avant de la jouer.
  — `removeJoueur()` : Supprime un joueur de la partie en cas d'abandon.
  — `carteJouable()` : Verifie si une carte peut être posee sur la table.
  — `dansMain()` : Verifie si une carte appartient encore a la main du joueur.
  — `botJouer()` : Implemente la logique de jeu d'un bot.

6 Pseudo-code du Système

Listing 1 – Pseudocode de la mecanique du jeu

```
>> DEBUT DU JEU

    TANT QUE aucun joueur n'a atteint 66 tetes clown OU X manches
       non jouees

       DEBUT DE MANCHE
            - Distribuer 10 cartes a chaque joueur
            - Placer 4 cartes initiales en ordre croissant sur la
               table

            TANT QUE les joueurs ont des cartes en main
                 - Chaque joueur pose une carte face cachee
                 - Reveler lorsque tous les joueurs ont choisi leurs
                    cartes ou le temps ecoule (30 sec)
                 - Placer les cartes une par une dans le rang ou il y
                    'a la plus petite diff positive par ordre
                    croissant
                 - Si la carte est la 6e d'une rangee, le joueur
                    prend les 5 cartes du rang (comptabilise comme
                    penalites )
                 - Si aucune rangee n'est possible, le joueur choisit
                     une rangee a recuperer et place sa carte dans la
                     premiere pos de la rangee
            FIN TANT QUE

       FIN DE MANCHE

    FIN TANT QUE

    Le(s) joueur(s) avec les moins de tetes clown gagne le jeu

>> FIN JEU
```

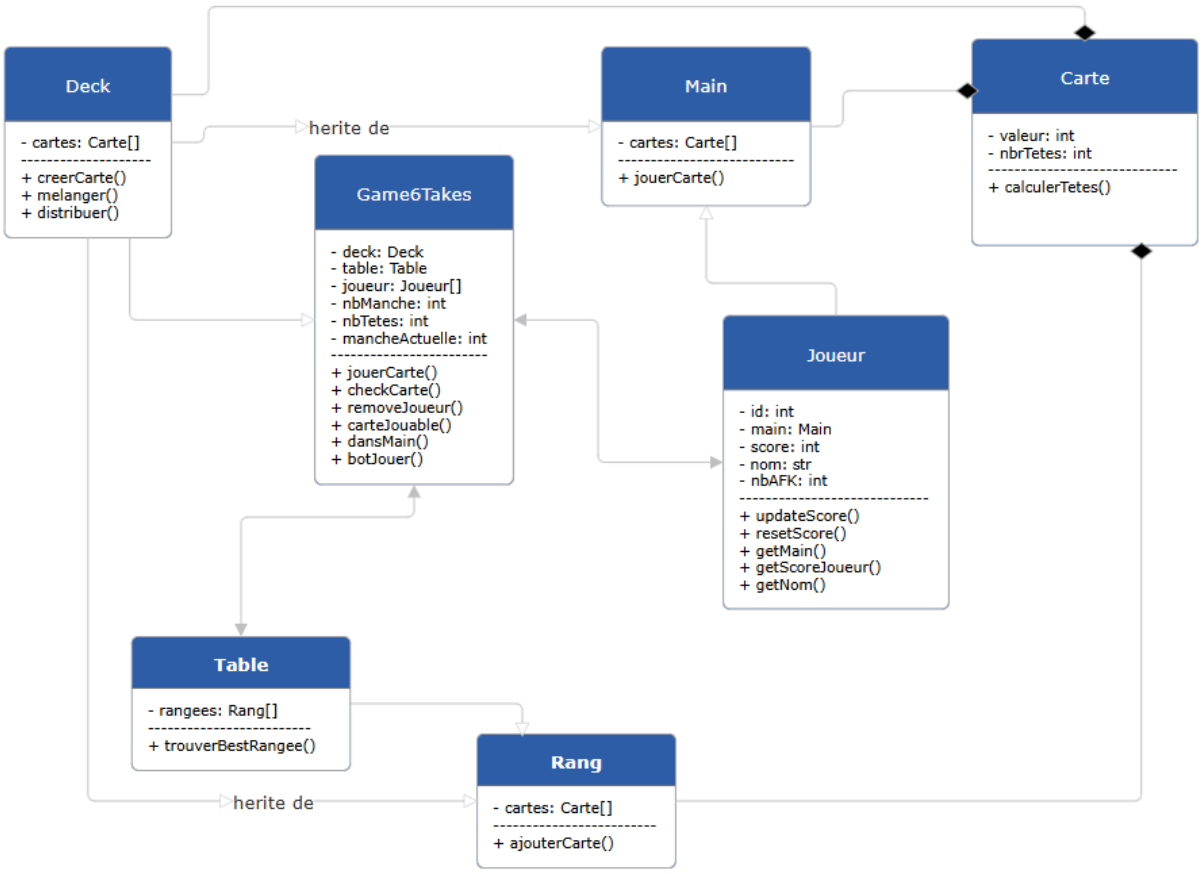


FIGURE 1 – Diagramme UML du jeu 6 qui prend !

7 Architecture du Serveur et Exigences Fonctionnelles

Utilisation d'Express.js :
- Express.js est utilisé pour gérer les requêtes HTTP.
- Il gère les actions non instantanées, comme l'inscription, la connexion et la récupération des données.
- Il repose sur des **routes API**, qui redirigent les requêtes vers des contrôleurs.

Utilisation de Socket.io :
- Socket.io est utilisé pour gérer les interactions en temps réel entre le client et le serveur.
- Il permet d'envoyer et de recevoir des événements instantanément (ex : jouer une carte, mise à jour du plateau).
- Contrairement à Express, il repose sur une connexion persistante avec le client.

7.1 Organisation des Routes API (Express.js)

Les routes API sont définies dans le dossier `routes/` et redirigent les requêtes vers les contrôleurs exemple :
- **routes/utilisateur.ts** : Gère l'inscription et la connexion des joueurs.
- **routes/partie.ts** : Gère la création et la gestion des parties.

7.2 Organisation des WebSockets (Socket.io)

Les événements WebSockets seront définis dans des contrôleur exemple : `socket.controller.ts`.
- **joinLobby** : Permet à un joueur de rejoindre une partie.
- **playCard** : Gère l'action de jouer une carte.
- **updateBoard** : Envoie une mise à jour du plateau à tous les joueurs.

8 Cas d'Utilisation

Connexion d'un Joueur :

CLIENT → DemandeConnexion(email, mdp) → [SERVEUR]
SERVEUR → Vérifie email/mdp avec la BDD.
BDD → Renvoie le résultat (succès ou échec).
SERVEUR → Renvoie 0 (succès) ou 1 (échec) → [CLIENT]

Un Joueur Rejoint une Lobby :

CLIENT → joinLobby(joueur, idLobby) → [SERVEUR]
SERVEUR → Vérifie si la partie existe et ajoute le joueur.
SERVEUR → Met à jour la base de données → [BDD]
BDD → Confirme l'ajout du joueur → [SERVEUR]
SERVEUR → Informe tous les joueurs → [CLIENTS]

Création d'un Lobby :

CLIENT → createLobby(joueur, nbJoueurs) → [SERVEUR]
SERVEUR → Génère un ID de partie.

SERVEUR → Stocke la partie en base de données → [BDD]
BDD → Confirme la création → [SERVEUR]
SERVEUR → Envoie l'ID au joueur → [CLIENT]

Un Joueur Quitte une Partie :

CLIENT → leaveGame(joueur, idLobby) → [SERVEUR]
SERVEUR → Supprime le joueur de la partie.
SERVEUR → Met à jour la base de données → [BDD]
BDD → Confirme la mise à jour → [SERVEUR]
SERVEUR → Informe les autres joueurs → [CLIENTS]

Mise à Jour du Plateau en Temps Réel :

CLIENT → playCard(joueur, carte) → [SERVEUR]
SERVEUR → Vérifie la validité de la carte (MainPartieController)
SERVEUR → Met à jour la base de données → [BDD]
BDD → Confirme l'action → [SERVEUR]
SERVEUR → Diffuse mise à jour du jeu à tous les clients → [CLIENTS]

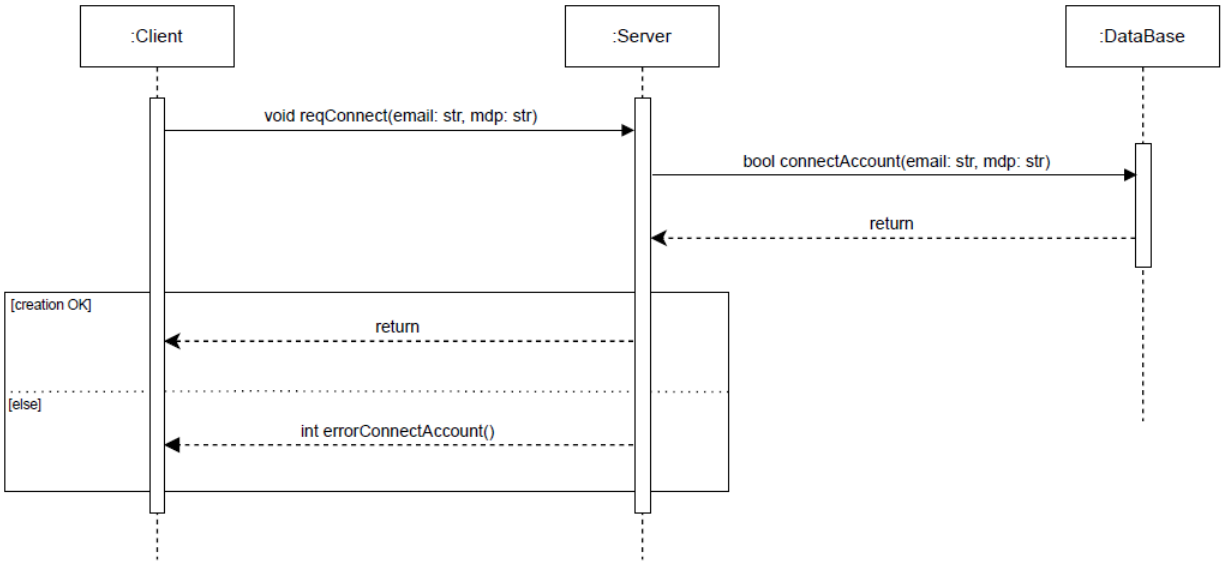
Édition du Profil d'un Joueur :

CLIENT → editProfile(idUtilisateur, newPseudo, newAvatar) → [SERVEUR]
SERVEUR → Met à jour les informations en base de données → [BDD]
BDD → Confirme la mise à jour → [SERVEUR]
SERVEUR → Retourne une confirmation de mise à jour → [CLIENT]

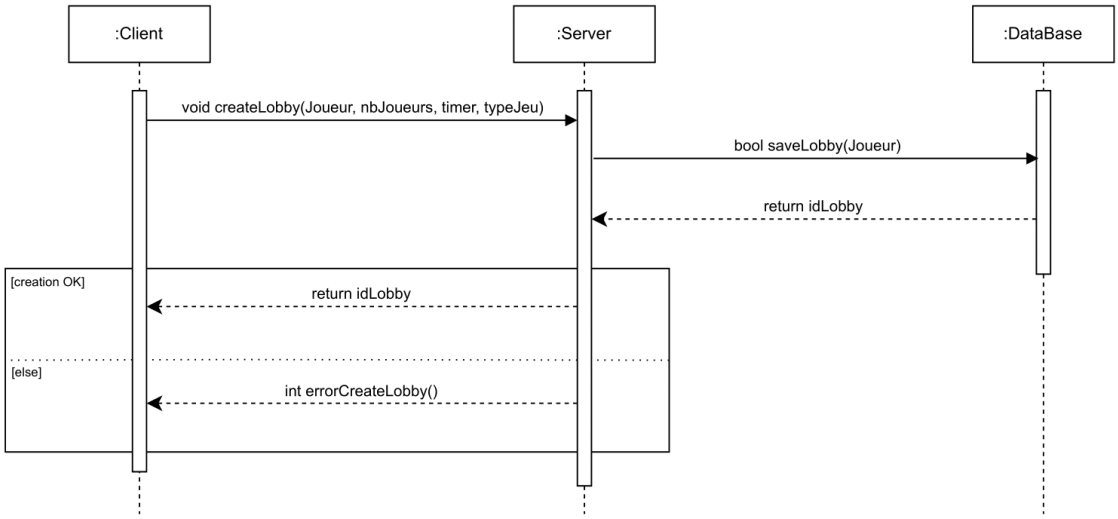
Déconnexion d'un Joueur :

CLIENT → logout(idUtilisateur) → [SERVEUR]
SERVEUR → Supprime le token de connexion.
SERVEUR → Informe les autres joueurs si la partie est en cours → [CLIENTS]

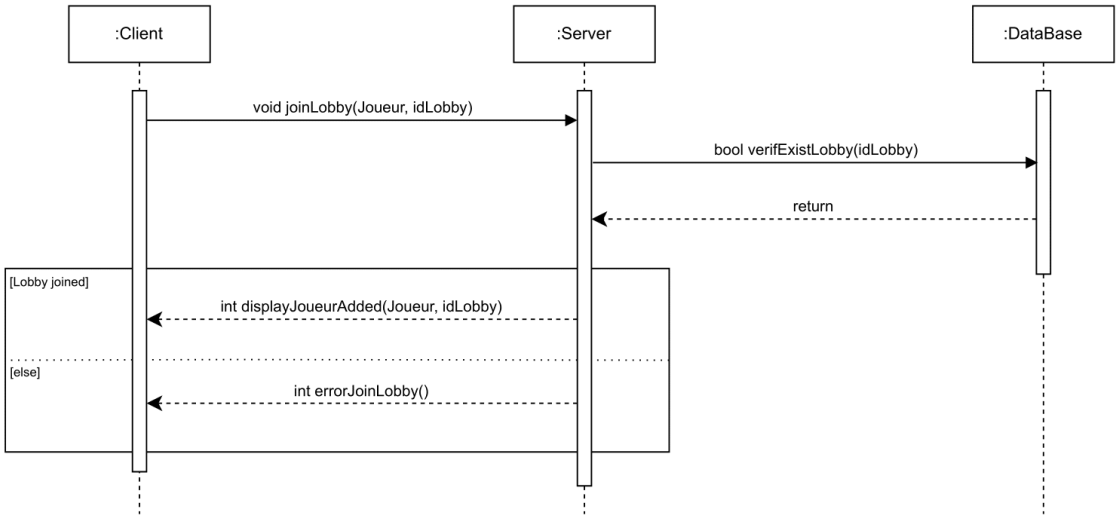
User connects to their account :



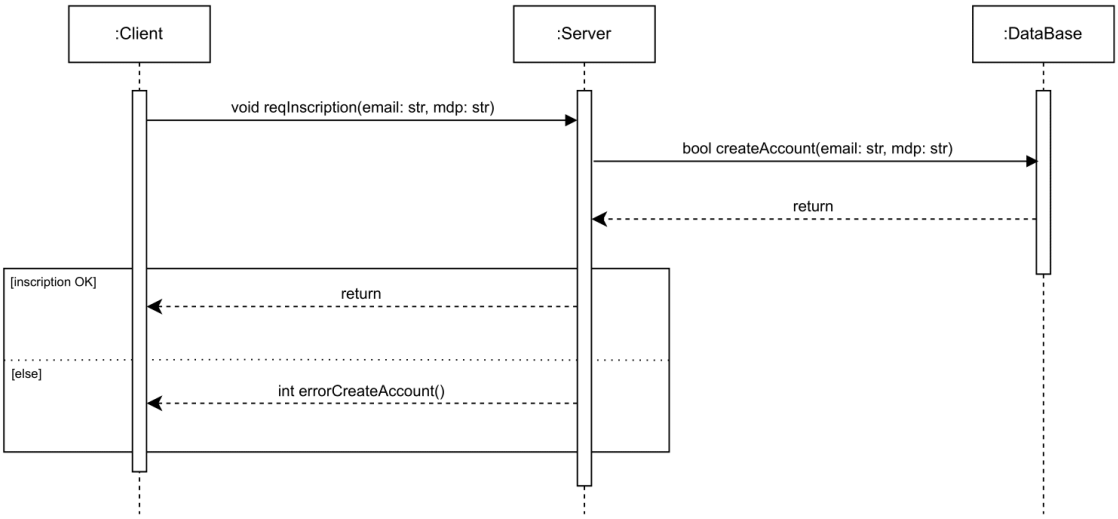
User creates a lobby :



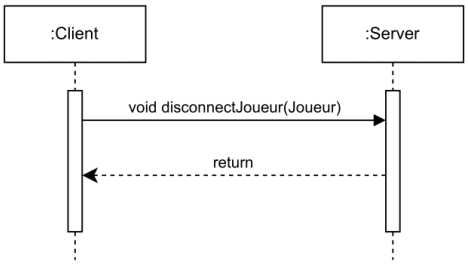
User joins a lobby :



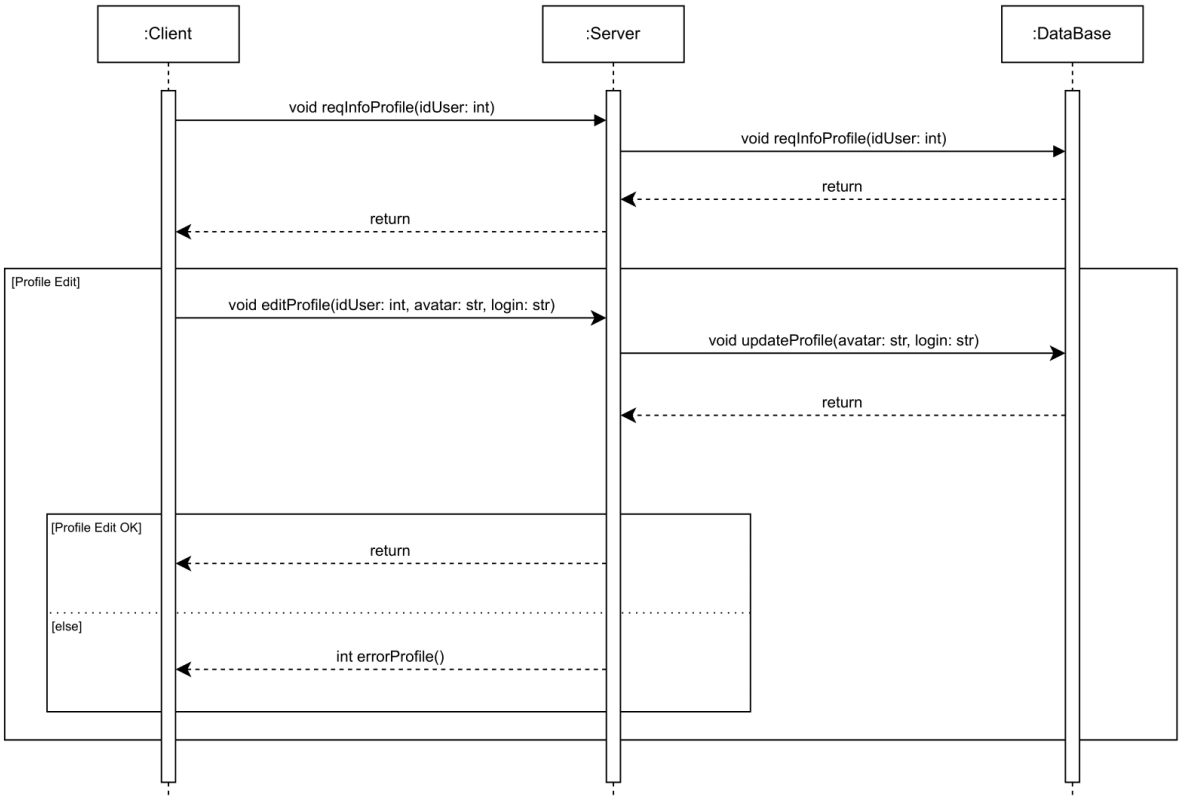
User creates an account:



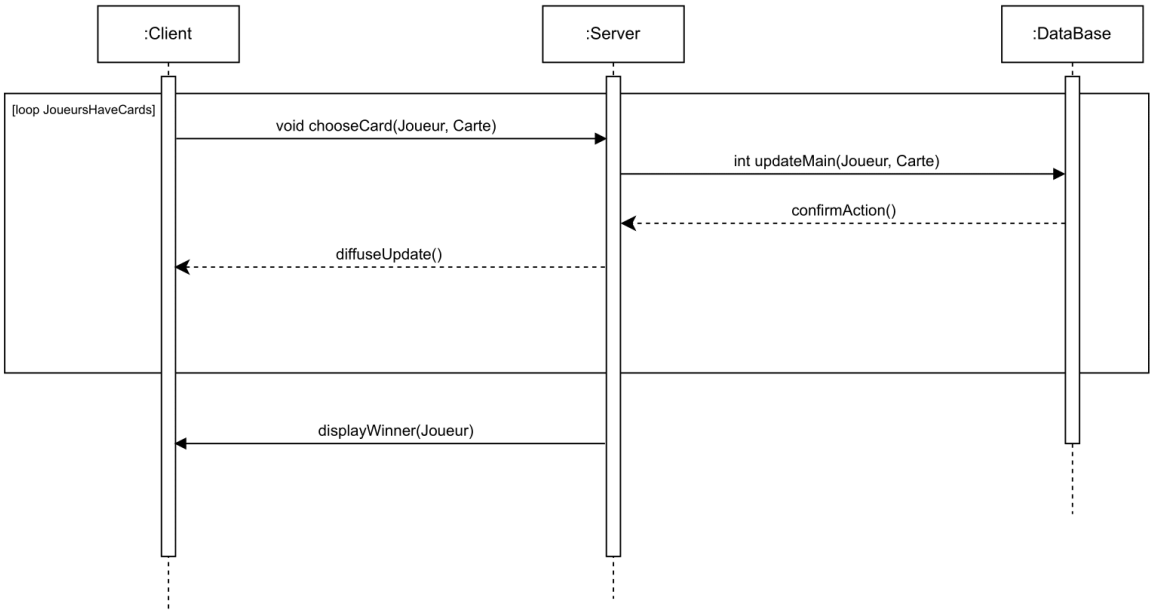
User disconnects from account :



User displays their profile:



Party Progression:



Joueur leaves the lobby/party: