Rapport – Jeu Saucisse

I. Choix techniques et implémentation

Nous avons choisi d'implémenter le Jeu Saucisse en utilisant une architecture réseau de type étoile, centralisée autour d'un serveur unique. Ce serveur joue un double rôle : il agit à la fois comme un proxy pour relayer les communications et comme un arbitre pour valider les actions de jeu. Cette architecture centralisée présente plusieurs avantages majeurs.

Premièrement, elle permet une validation centralisée des coups. Le serveur, via le fichier *serverB.py*, utilise les méthodes *ValidateSausage* et *CheckCrossing* pour vérifier systématiquement chaque saucisse placée par les joueurs. Ces vérifications portent sur les distances entre les points (maximum 2 cases en ligne et en colonne) et sur les éventuels croisements entre saucisses. Cette approche garantit une parfaite cohérence du jeu entre tous les clients connectés.

Deuxièmement, le serveur assure une gestion unifiée des règles du jeu. Il prend en charge le calcul des scores ELO selon le système décrit dans Moodle, gère le système d'invitations entre joueurs et détecte les fins de partie via la méthode *check_end_game*. Cette centralisation des règles garantit l'équité entre tous les joueurs.

Enfin, notre implémentation offre une bonne tolérance aux déconnexions. Lorsqu'un joueur quitte une partie, le serveur notifie immédiatement son adversaire via *Network_opponent_disconnected* et réinitialise proprement le lobby. Pour optimiser les performances, nous avons également implémenté une pré-validation côté client avec la méthode *validate_local_sausage*, ce qui réduit le nombre de messages envoyés au serveur.

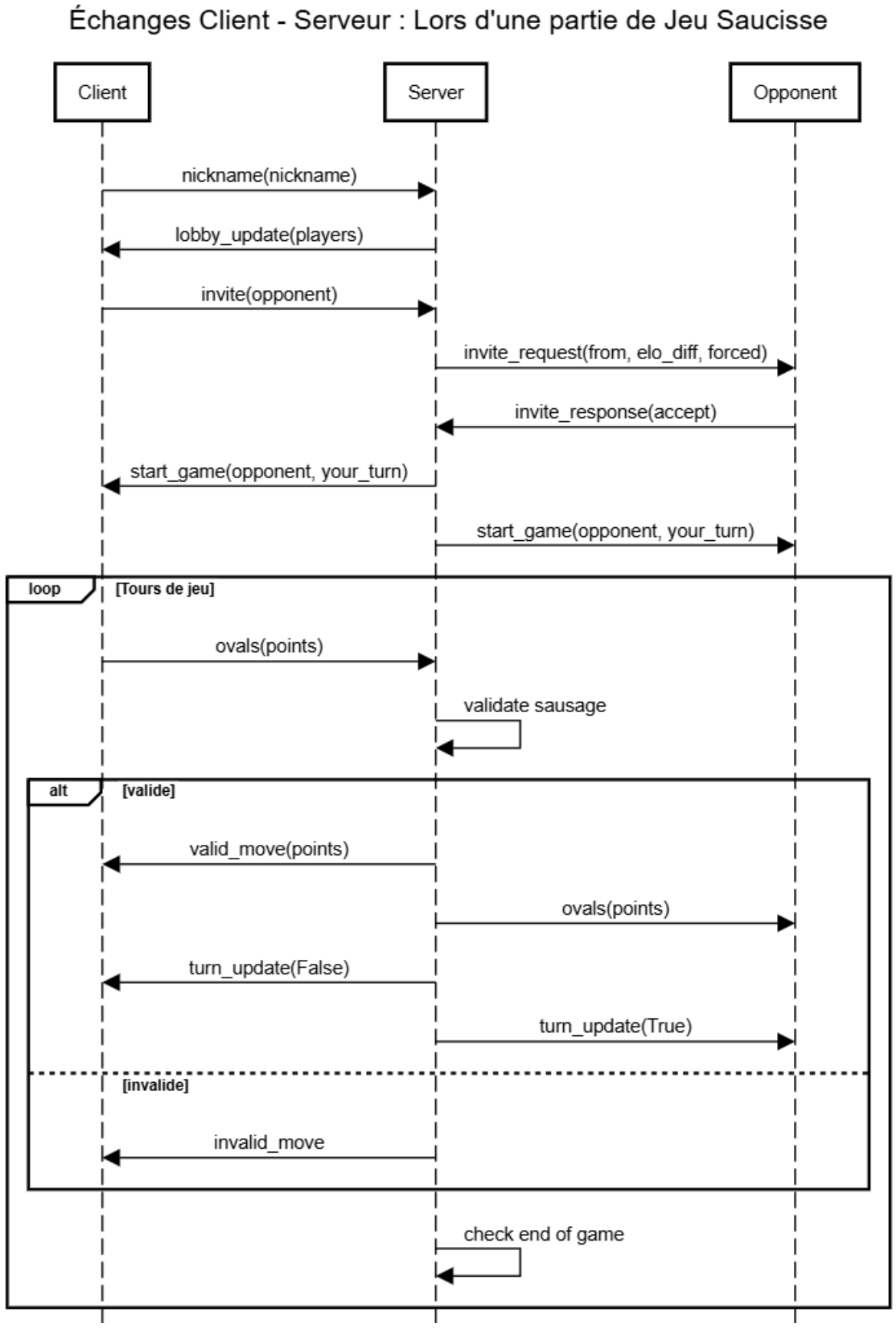
Concernant les règles spécifiques du jeu, nous avons opté pour une interprétation stricte des consignes du premier cours. Une saucisse n'est valide que si tous ses points sont espacés d'au maximum 2 de tous les autres points de la saucisse. Contrairement aux règles indiquées sur Moodle qui demandent l’existence d’au moins 1 (et non de tous) point à une distance inférieure ou égale à 2 en ligne et en colonne.

Pour la gestion de notre code source, nous avons utilisé GitHub comme système de versioning. Cette plateforme nous a été particulièrement utile lorsqu'un fichier a été accidentellement écrasé, nous permettant de restaurer rapidement une version fonctionnelle du projet.

II. Organisation

Pour implémenter notre jeu nous nous sommes d’abord un peu perdus avec les appels à réaliser entre client et serveur. Nous avons donc posé sur feuille un schéma des échanges à

réaliser entre les deux fichiers *serverB.py* et *clientB.py*.



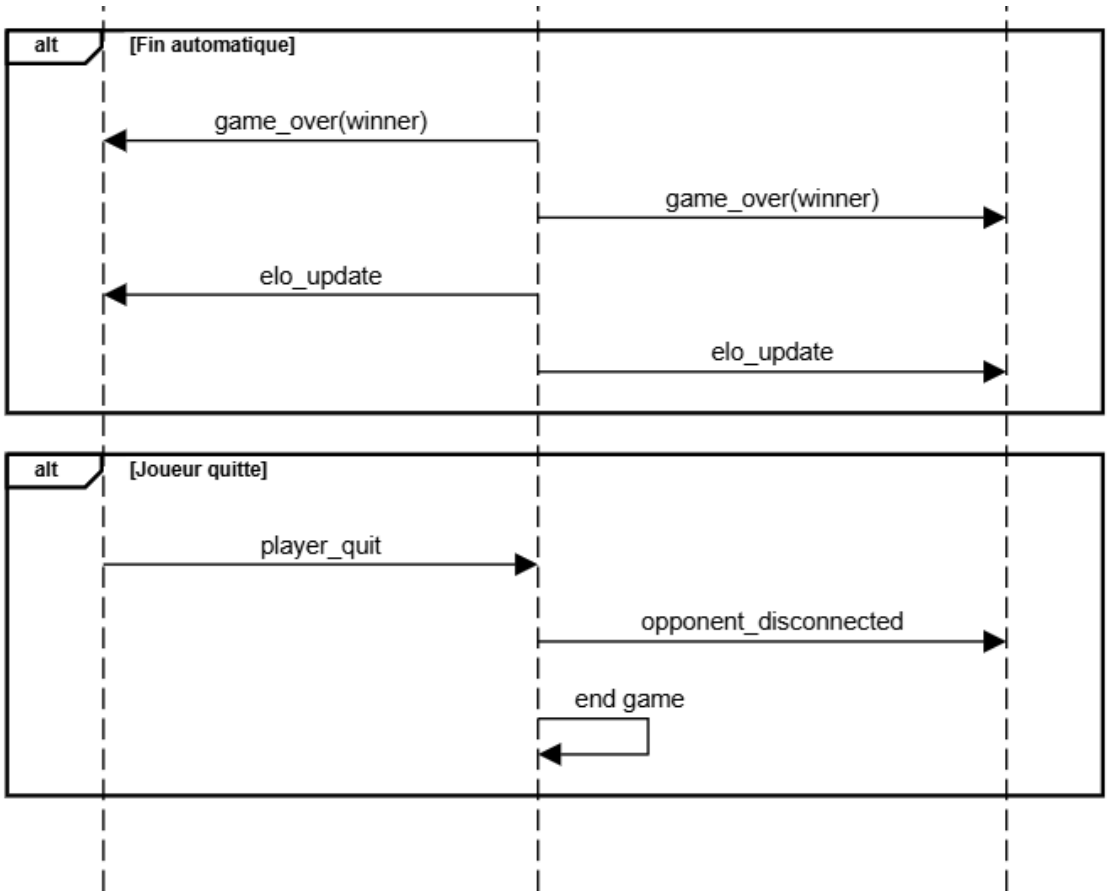


Schéma de séquence réalisé avec sequencediagram.org

Nous nous sommes tous les trois basés sur ce schéma et l'avons testé sur plusieurs ordinateurs régulièrement pour vérifier le bon fonctionnement des modifications apportées.

III. Difficultés et idées d'améliorations

Notre développement s'est heurté à plusieurs difficultés techniques. La principale contrainte vient du module PodSixNet que nous utilisons pour la communication réseau. Cette bibliothèque, bien que pratique, est une ancienne version uniquement compatible avec Python 3.11. Cette limitation nous a obligés à travailler principalement sur les machines du CREMI, l'utilisation de X2Go Client s'étant avérée peu pratique en raison des exigences en bande passante (connexion).

Un autre problème important concerne la robustesse du serveur. Actuellement, si le serveur principal rencontre un problème et s'arrête, tous les joueurs sont immédiatement déconnectés sans possibilité de reprendre leurs parties en cours. Une amélioration évidente serait d'implémenter un système de serveur de secours.

La duplication de code entre les validations côté client (*validate_local_sausage*) et côté serveur (*ValidateSausage*) constitue également une limitation de notre implémentation actuelle. Bien que cette approche améliore les performances en réduisant les appels réseau, elle crée une redondance dans le code et dans la vérification des règles.

La gestion des pseudos des joueurs présente aussi des faiblesses. Notre système actuel permet à deux joueurs d'avoir le même pseudo, ce qui peut créer des conflits (pas d’affrontement possible). Nous avions envisagé de mettre en place un système complet d'enregistrement avec pseudos et mots de passe, qui aurait également permis de sauvegarder les scores ELO entre les sessions. Cependant, par souci de simplicité et par manque de temps, nous n'avons pas implémenté cette fonctionnalité.

Parmi les améliorations possibles, l'ajout d'un chat intégré au jeu permettrait aux joueurs de communiquer. Un tel système devrait inclure des mécanismes de modération pour prévenir les comportements inappropriés, comme les insultes ou les menaces, courants dans les jeux en ligne.

Enfin, une fonctionnalité plus ambitieuse consisterait à ajouter un système d'analyse statistique en fin de partie. Cet outil pourrait par exemple identifier les coups clés qui ont influencé l'issue de la partie. Cependant, la mise en œuvre d'une telle fonctionnalité s'avère nettement plus complexe et dépassait le cadre de notre projet actuel.

Un problème qui arrive seulement dans des conditions rares est qu’un coup valide peut être indiqué comme étant non valide alors qu’il l’est. Ceci est dû à l’ordre de sélection des points. Il suffit au client de réessayer dans un ordre différent.