Toutes les informations dans ce document sont sujets à changement

Questions pour la séance:

-Combien de sockets comment gerer chat/partie/etc

-Mode spectateur?

-(facultatif) gerer reconnexions ou pause pour remettre à plus tard?

**Le serveur:**

**Introduction:**

Le serveur a plusieurs fonctions: il doit pouvoir gérer les interactions entres amis, gérer la recherche de parties/d'adversaires, fournir un système de classement, fournir un système de matchmaking et sa fonction principale: il doit gérer des parties d'échecs de 2 (voir plus selon les modes) clients.

**Communication client-serveur dans le cas d'une partie:**

Limitons nous à la description d'un jeu d'échec aux règles classiques dans un premier temps:

Dans le cadre d'une partie le client communique les informations suivantes au serveur:

-Un tuple (index\_pion\_a\_bouger, (x\_2, y\_2)) contenant l'index de la pièce à bouger qui doit être le même sur le client et serveur et la position où le client souhaite bouger cette pièce.\*

Le serveur se charge de vérifier si cette action est possible dans le cas positif:

1)ce sera au tour de l'autre client il va donc prévenir le client qui vient de jouer que son tour a été validé.

2)Envoie à l'autre client des changements\*\* de positions éffectués lors du tour de l'autre client ainsi qu'une notification pour lui prévenir que c'est son tour de jouer.

Dans le cas négatif: le client devra jouer à nouveau et devra encore envoyer le tuple jusqu'à ce qu'il envoie un tuple qui est validé par le serveu puis ce sera retour au point 1) dans le cas où l'action est validée.

\*(alternativement ancienne pos, nouvelle pos)

\*\*il peut y avoir plusieurs changements par tour ex pièce qui meurt

Diagramme de séquence:

**Implémentation du système de classement:**

Différent classements pour chaque mode de jeu et un classement global?

A la fin de chaque partie on ajoute une victoire/défaite/match null (si il y en a aux échecs) aux joueurs de la partie puis on met à jour le classement (en prenant en compte qu'il peut y avoir d'autres parties qui se finissent qui sont en train de modifier le classement aussi).

**Implémentation du matchmaking:**

Pré-requis: définir une métrique d pour pouvoir sélectoinner un adversaire (ex: ratio, nombres de victoires, nombre de points où on gagne 1 point par victoire et perds 1 point par défaite, par experience etc...) cette fonction métrique renvoie un nombre devrait pouvoir mesurer la différence de niveau entre chaque joueur.

Lorsqu'un client désire chercher un adversaire il le signale au serveur qui va procéder de la manière suivante (j est le joueur qui cherche une partie):

Algo (todo expliquer ca plus proprement):

Soit l'ensemble E de la listes des joueurs qui cherchent une partie.

Pour tous les j' appartenant à E si il y a d(j, j') < 1 alors on place le joueur contre j' .

Si il n'a pas de j' on rajoute le joueur à la liste et on attend 15 secondes.

Si après ces 15 secondes il n'a pas eu un nouveau client j' qui s'est connecté avec un d(j, j'à plus petit on le place avec l'adversaire j' où d(j, j') est le plus petit si il n'y a aucun j' connectés au serveur alors on attend jusqu'à ce qu'un adversaire ce connecte.

Une fois que des clients ont trouvé une partie on crée un thread ?

**Communication client-serveur dans le cas d'interaction entre amis:**

-Possibilité de supprimer/ajouter des amis

-Possibilité d'inviter des amis/rejoindre des amis pour jouer avec eux

-Possibilité d'envoyer/recevoir des messages entre amis

Besoin non fonctoinnels:

Serveur doit être stable (client aussi)

**Idées d'implémentation:**

Le premier byte (unsigned int?) décrit le type du message et le serveur/client ont des fonctions prévues pour prendre chaque message en compte.

JE DOIS JUSTE DIRE COMMENT LE SERVEUR REPONDS AU REQUETES FAIRE