Jeu Streams - Analyse de l'application

Infos administratives

Numéro de groupe: 15

Etudiant 1 : Debin Pierre-Alexandre

Etudiant 2 : Devos Thibaut

Etudiant 3 : Massart Xavier

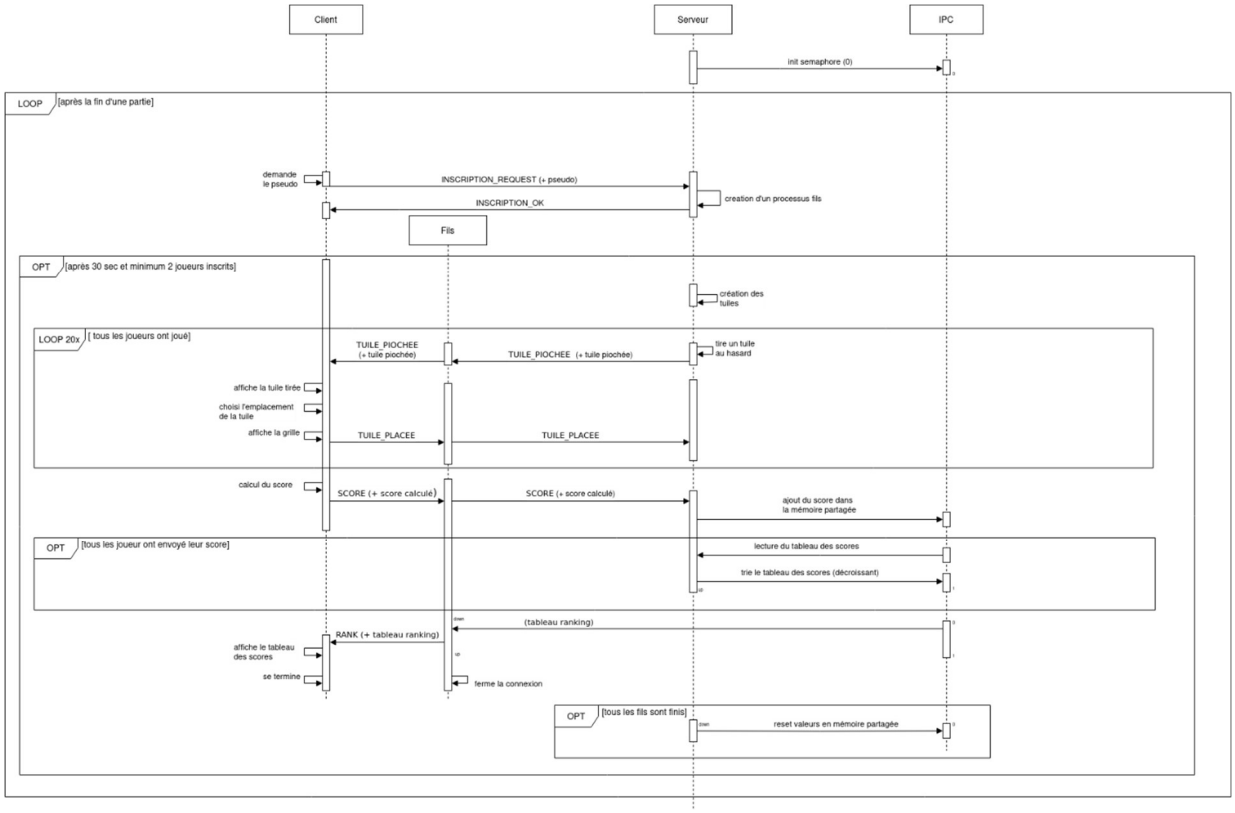
Communication

Donnez ici le type des messages échangés (sur les pipes et les sockets).
Donnez les scénarios d'exécution.

Messages échangés :

INSCRIPTION_REQUETE
INSCRIPTION_OK
INSCRIPTION_KO
TUILE_PIOCHEE (server -> fils -> client : envoie de la tuile piochée)
TUILE_PLACEE (client -> fils : indique qu'il a placé la tuile)
SCORE (client -> fils -> serveur : envoie du score du client)
RANK (fils -> client : envoie du leaderboard)

Scénario exécution :



Découpe de l'application

L'application sera découpée de la manière suivante :

Serveur :

- Inscription (gère l'inscription, crée un fils, utilise network)
- Boucle de jeux (Hazard de tuile et distribution)
- Fin de partie (trier les scores, envoie leaderboard en mémoire partagée)

Fils :

- Fait le pont entre le père (serveur) et le client
- Distribue les tuiles au client reçues par le serveur
- Récupère la confirmation de jeux et l'envoie au serveur
- Fin de partie (récupère leaderboard sur mémoire partagée + envoie client)

Client :

- Inscription (demande d'inscription)
- Boucle de jeux (placement / affichage / envoie de confirmation)
- Calcule ses points et envoie au fils
- Fin de partie (affichage le leaderboard)

Modules

Donnez ici le nom des modules et une brève description.
Donnez les signatures des fonctions et une brève description de ces fonctions.

Network : (comportera tout ce qui permet de faire passer/recevoir des messages sur le réseau)

Signature	Description	Serveur/Client
void initSocketClient(char *serverIP, int serverPort)	Permettra d'initialiser le socket coté client (Crée le socket et se connecte au serveur) renvoie le socket	Client
void initSocketServer(int port)	Permettra d'initialiser le socket coté serveur (Crée le socket, bind(réservation d'un port), listen(=> socket passif) renvoie le socket	Serveur
void closeSockets(Player*tabPlayer, int nbPlayer)	Fermera les connections client	Serveur
void closeClientSocket()	Ferma la connexion server	Client

IPC : Permettra la gestion d'accès à la mémoire partager (tout en utilisant les sémaphores)

Signature	Description	Serveur/Client
Int* getAllScores()	Fournira un tableau des scores NON trié	Serveur

	Indice = tableau des players	
void ajouterScore(int player, int score)	Ajoute le score du « joueur » avec le « score »	Server
void trierScore()	Permet de trier les score dans l'ordre décroissent	Server
void cleanIpc()	Fermer tous les ipcs	Server

JEU : Contiendra toute la logique du jeux (Hazard tuile, compté les suite)

Signature	Description	Serveur/Client
void jeux()	(Représentera le joueur responsable de la pioche) Tire la tuile puis envoie au fils qui envoie au client	Serveur
Int hazardTile(int* tiles)	Génèrera une tuile aléatoire parmi la table (quand une tuile est prise la valeur sera remplacer pars -1)	Serveur
Int comptePoint(int* gamePaper)	Calculera les points lors que l'on lui donne une feuille de jeux	Client
void placeTuile(int index)	Placera une tuile a l'indice enter (si déjà pris sera placer à droite)	Client

Types Utilisateurs

Donnez ici les types utilisateurs que vous comptez créer et une brève description.

Structure d'un message :

- **Enuméré code**
- **Variables :**
  pseudo *(char[taillePseudo])*
  tuilePiochee *(int)*
  scoreJoueur *(int)*
  tableauJoueurs *(struct) shm*

Enuméré code :

INSCRIPTION_REQUETE
INSCRIPTION_OK
INSCRIPTION_KO
TUILE_PIOCHEE *(server -> fils -> client : envoie de la tuile piochée)*
TUILE_PLACEE *(client -> fils : indique qu'il a placé la tuile)*
SCORE *(client -> fils -> serveur : envoie du score du client)*
RANK *(fils -> client : envoie du leaderboard)*

Structure d'un client :

- **Variables :**
  sockfd *(int)*
  pipe *(int)*
  pid *(int)*

Structure d'un joueur :

- **Variables :**
  pseudo *(char [])*
  score *(int)*

Structure tableau de joueur (shm) :

- **Variables :**
  tableauJoueurs *(struct joueur[MAX_JOUEUR])*
  nbDeJoueurs (int)