Groupe : A05

Allemand Valentin
Caucal Thomas

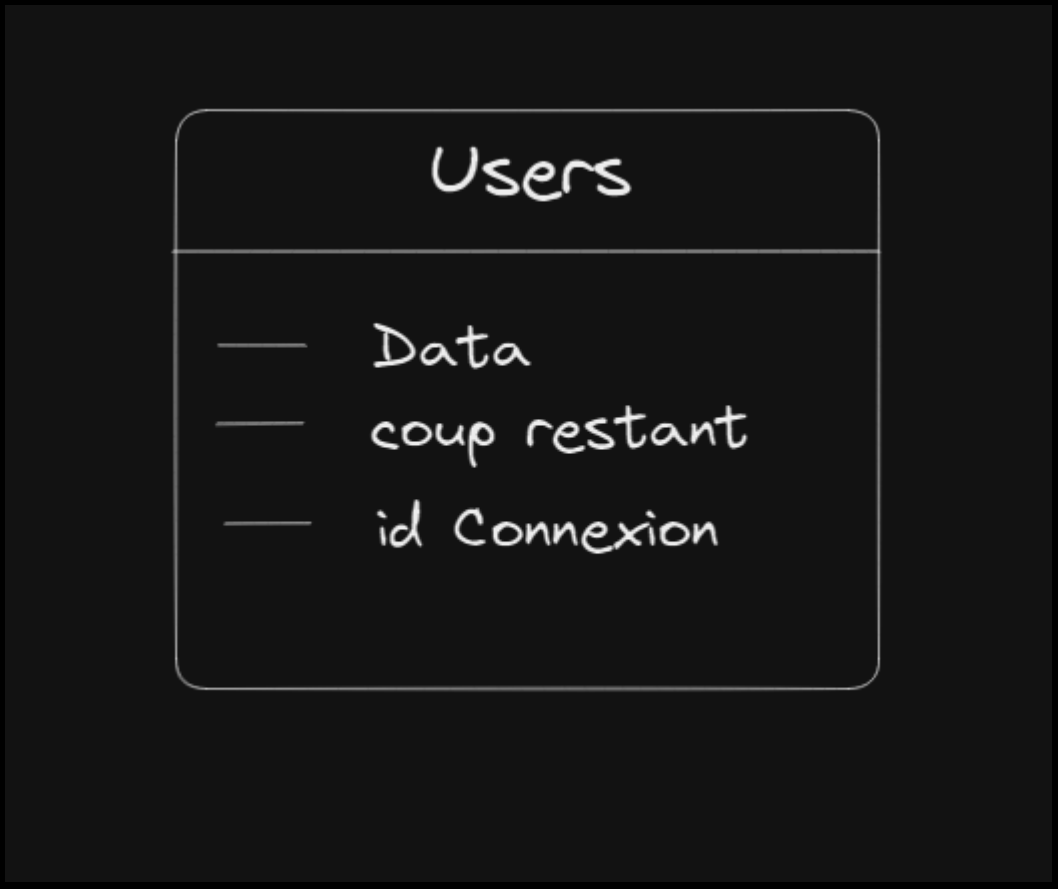
Document d'architecture

Projet: PixelWar

1° Les utilisateurs

A°) Structure Users

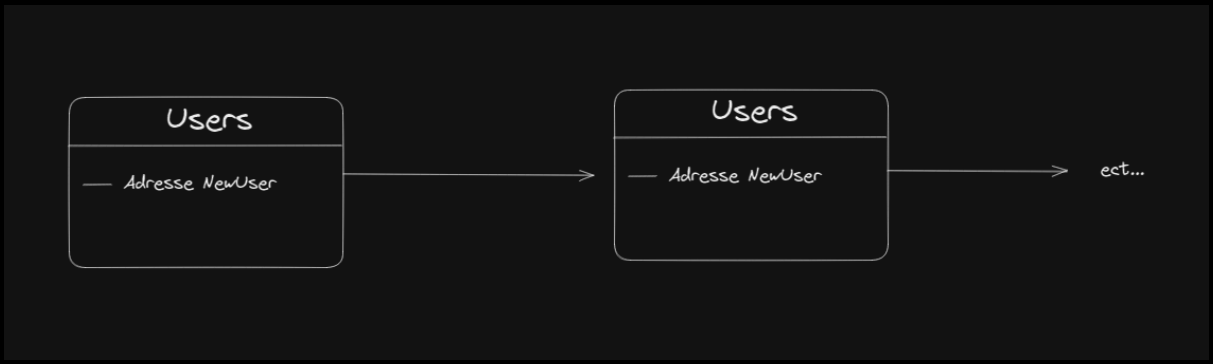
La première partie du projet à été de penser à la façon de gérer les utilisateurs sur le serveur. Pour ce faire, nous avons pensé à utiliser une structure spécialement dédiée aux gens voulant jouer au pixel war:



Nous avons choisi de mettre dans la structure User plusieurs int comme les coups restants pour le jouer , l'id de connexion ainsi que des datas diverses. Nous verrons par la suite que cette déclaration sera modifiée.

B°) Liste chaînée

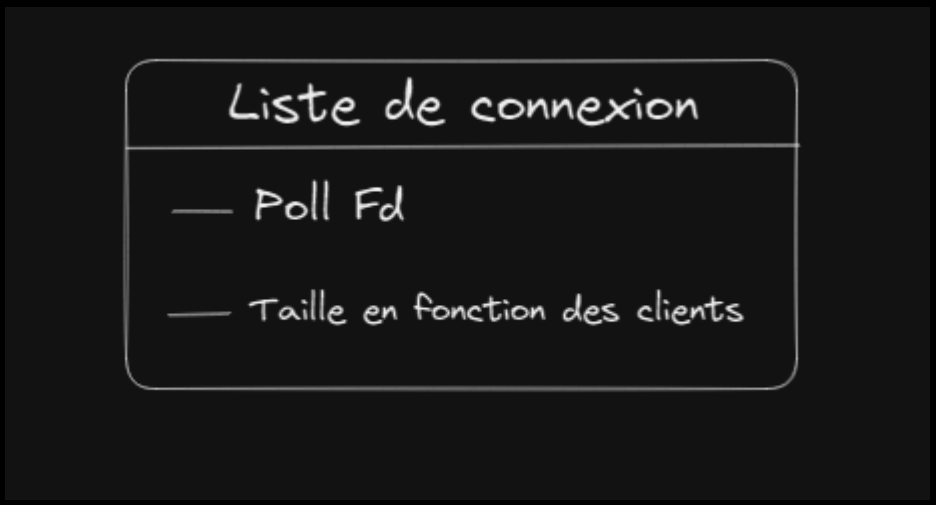
Pour organiser ces users, on utilisera une liste chainée (une file) , c'est à dire que chaque user pointera vers un autre user ce qui formera ainsi une file :



2° Gestion avec Poll

A°) Initialisation

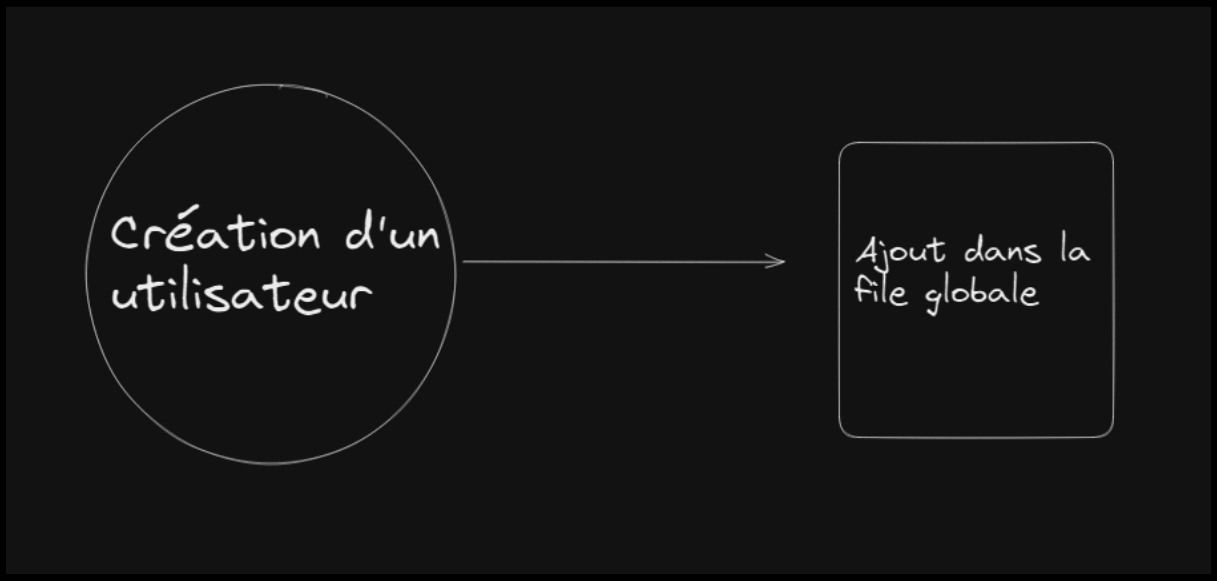
Pour commencer, il faut initialiser les différentes structures pour le Poll. On commence par créer un tableau qui contiendra tous les sockets des clients qu'il faudra écouter.



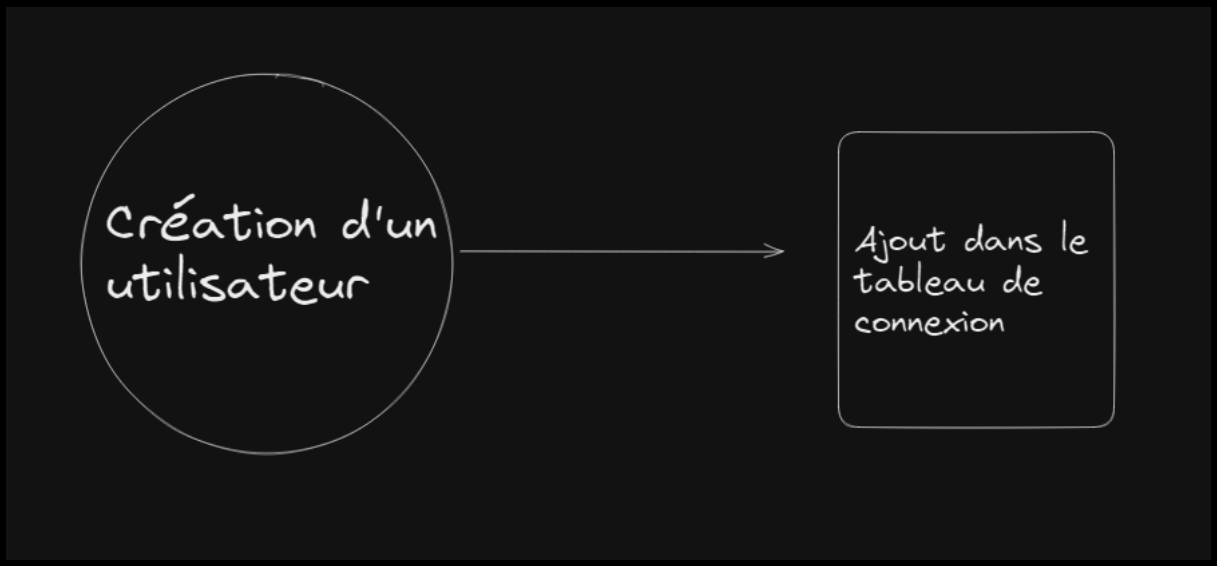
On initialise le premier socket avec le socket d'écoute du serveur (défini lors du lancement) et on met l'élément "events" à POLLIN (pour recevoir les messages)

B°) Détection d'actions

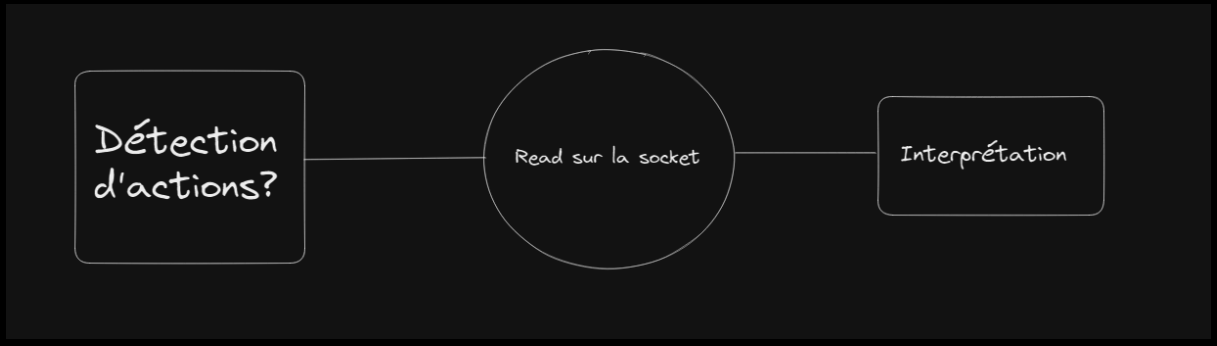
Pour détecter les actions, on parcourt le tableau des users en poll pour voir si un socket à changé d'état. Si le premier socket détecte une connexion grâce aux revents,on crée un nouvelle utilisateur que l'on ajouter à une file globale:



On n'oublie pas également d'accepter la connexion pour le poll et d'ajouter le nouvel utilisateur dans le tableau des listes de connexion.

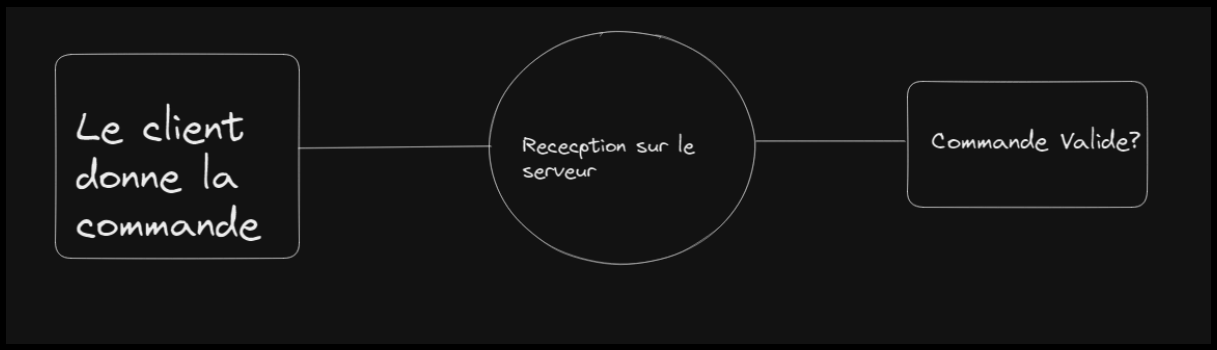


De la même façon, si une action est détectée sur les sockets d'écoute on la prend en compte en utilisant la fonction "read" et on interprète la demande:



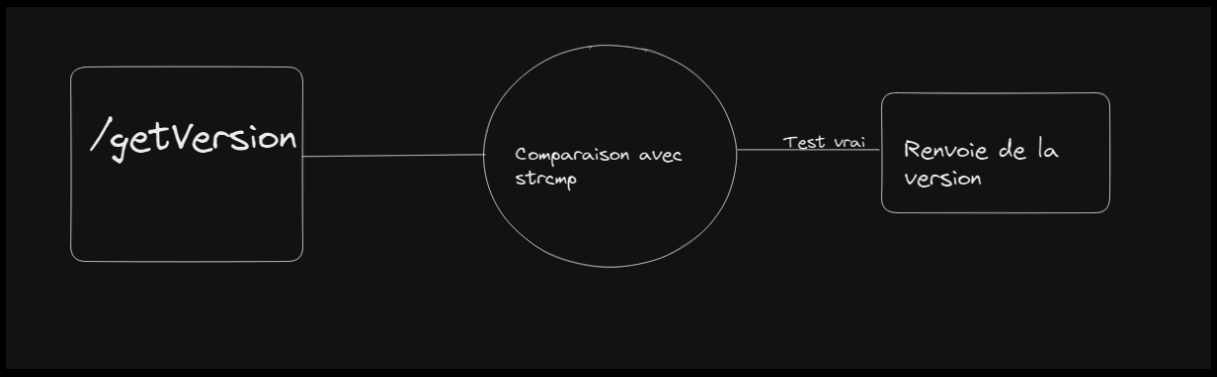
3° Découpage et Interprétation des commandes

Une fois le client connecté, il faut pouvoir interpréter ses actions entre autres, les commandes qu'il va taper.



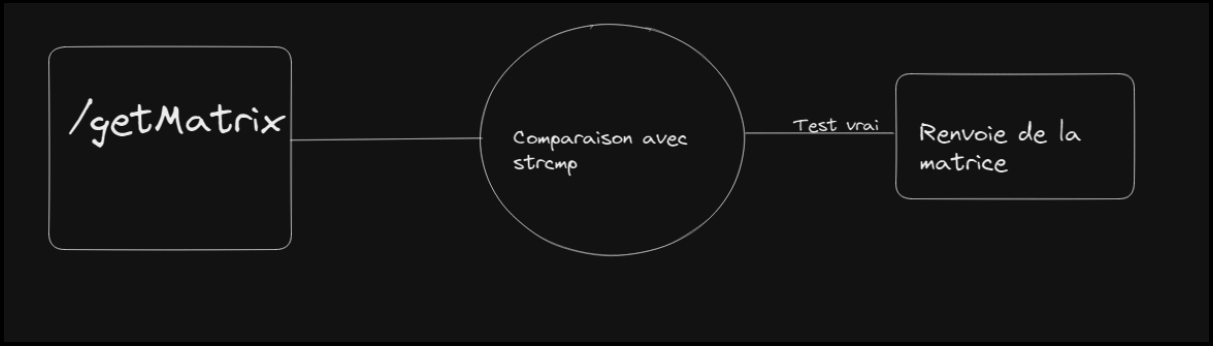
A°) Demande de la version

Tout d'abord on peut tester la commande tapée si c'est une commande qui ne nécessite aucun paramètre comme pour la demande de version par exemple. On utilise strcmp pour voir si les 2 chaînes de caractères sont égales:



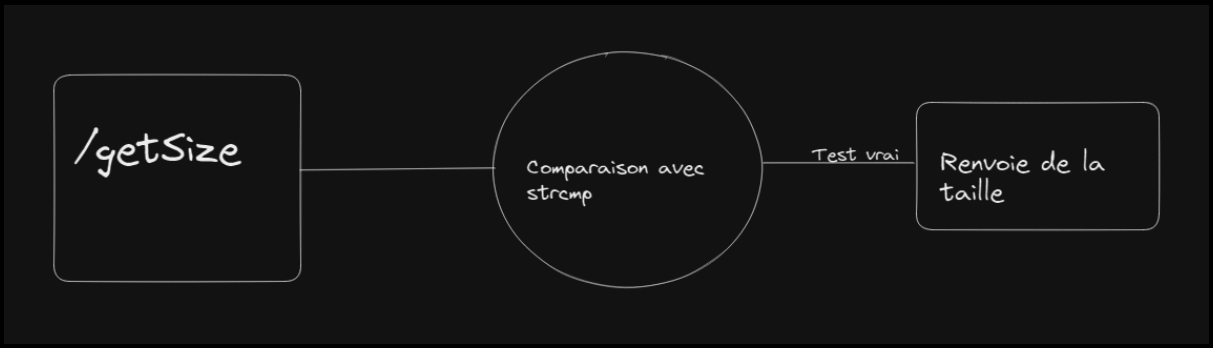
B°) Cas du renvoi de matrice

De la même façon, on peut également gérer la demande de renvoie de matrice :



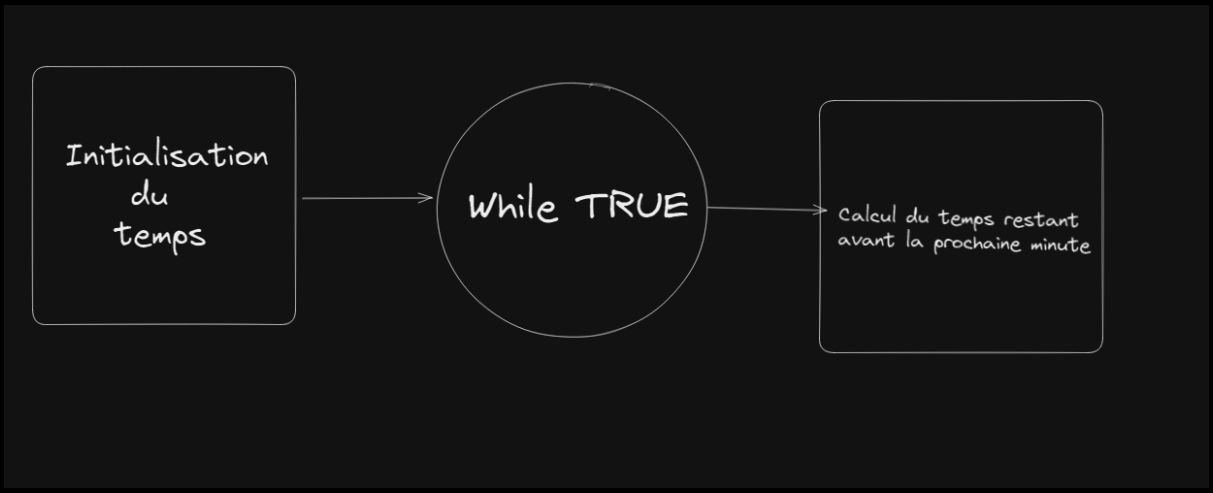
C°) Cas du renvoi de la taille

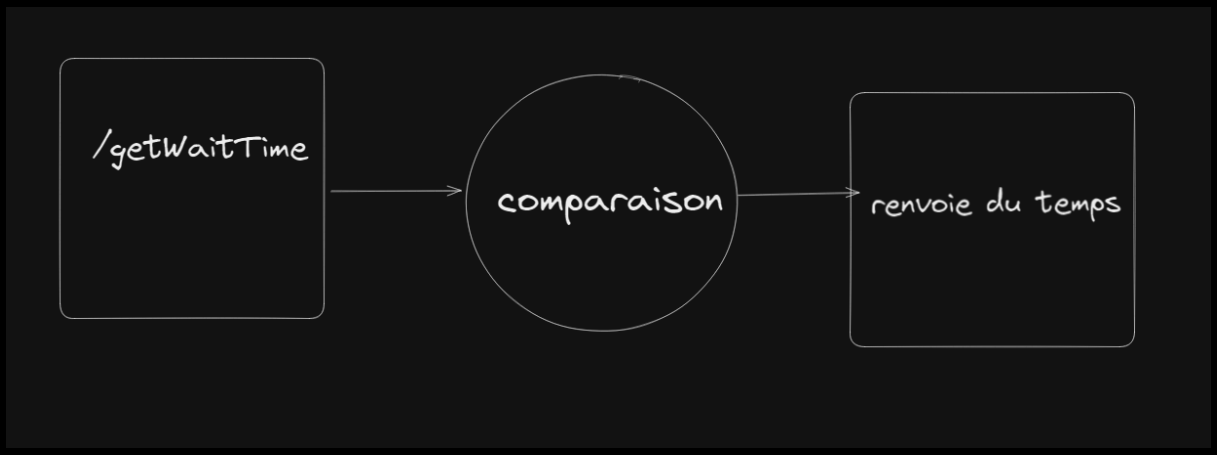
Encore une fois , on peut également gérer la demande de renvoie de taille :



D°) Cas du renvoi du temps

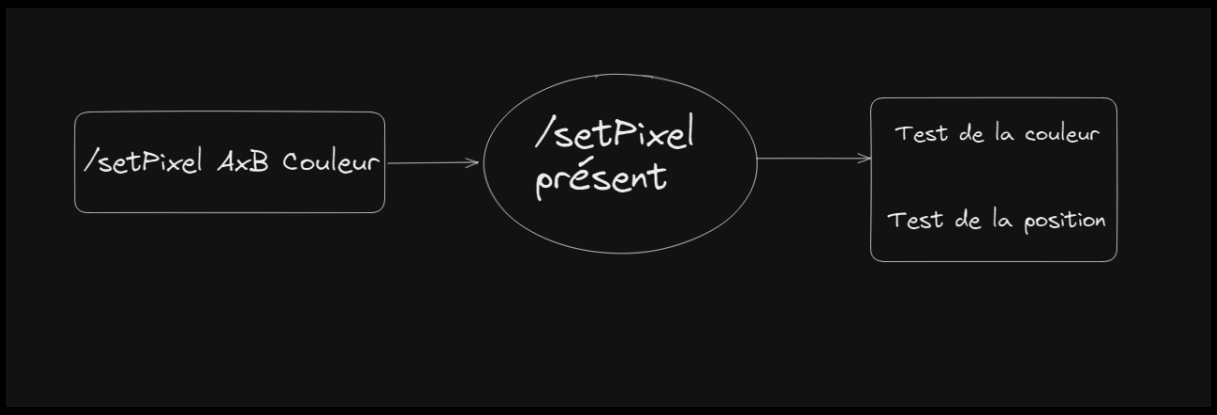
Pour renvoyer le temps, nous allons d'abord expliquer comment celui-ci marche sur le serveur. Lors du lancement du serveur, on récupère le temps en secondes depuis le lancement de UNIX. Ainsi, lorsque l'on est dans la boucle while, il nous suffit de récupérer l'heure actuelle via les fonctions de time.h et de calculer la différence entre les 2, si le nombre est plus petit qu'une minute, alors on ne fait rien. Lorsque 1 minutes de plus c'est écoulé dans la soustraction, on remet tous les coups restants des users à leurs valeurs de départ.





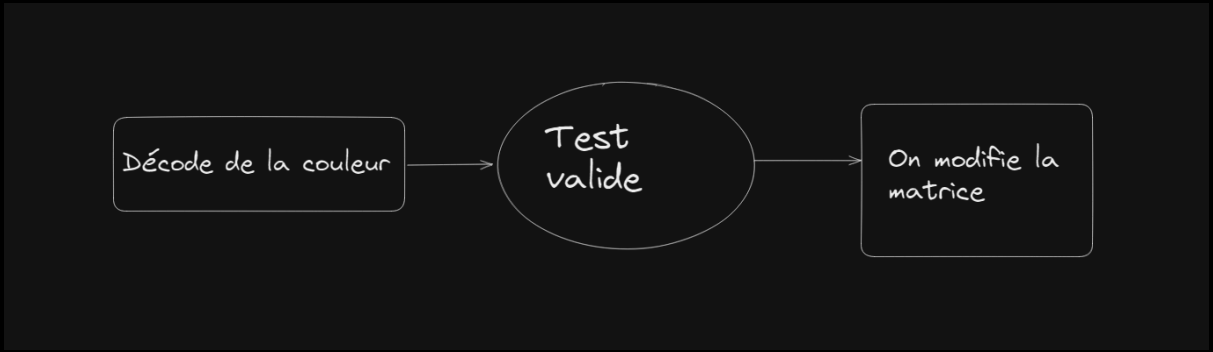
E°) Cas du setPixel

La commande la plus difficile à interpréter est celle du /setPixel. En effet , celle-ci prend en argument 2 choses, la position du pixel a modifié ainsi que la nouvelle couleur, il faut donc le prendre en compte dans l'interprétation de la commande. Tout d'abord il faut reconnaître le /setPixel car si celui-ci n'est pas reconnu, alors le serveur peut retourner l'erreur 99 unknown command



Pour tester la position, il suffit de voir si les dimensions choisies sont compatibles avec la taille actuelle de la matrice .

Pour tester la couleur c'est plus compliqué. En effet, on reçoit celle-ci encodé en base64, la première étape est donc de la décoder et ensuite on compare les couleurs reçues en RGB pour voir si elles sont valides



F°) Commandes inconnues

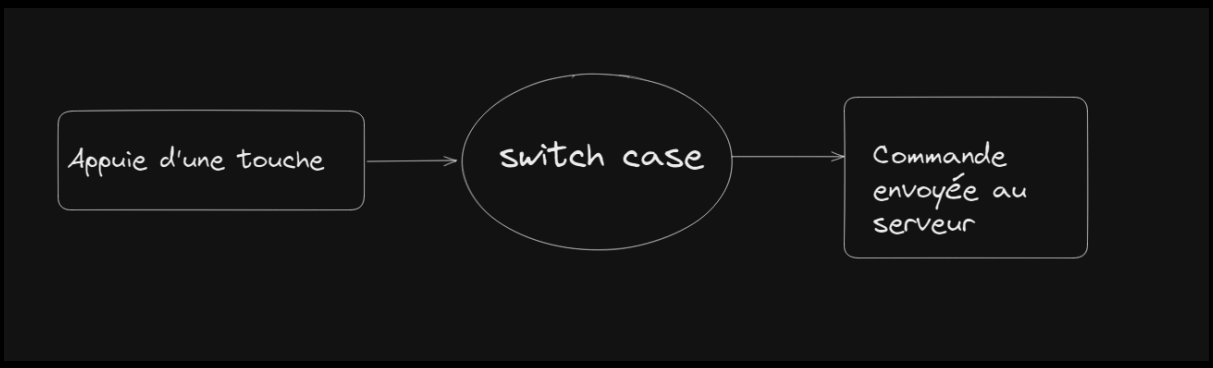
Si aucun de ces cas n'est vérifié, la commande n'est pas prise en charge par le serveur, il renvoie l'erreur 99.

5° Clients

Pour faciliter le jeu pour les utilisateurs, nous avons mis en place un client qui évite de taper toutes les commandes pour jouer. En effet, le client permet de mettre en place un menu qui est affiché à chaque itération après la réponse du serveur. De plus, celui-ci est nécessaire pour communiquer en base64 avec le serveur (cahier des charges).

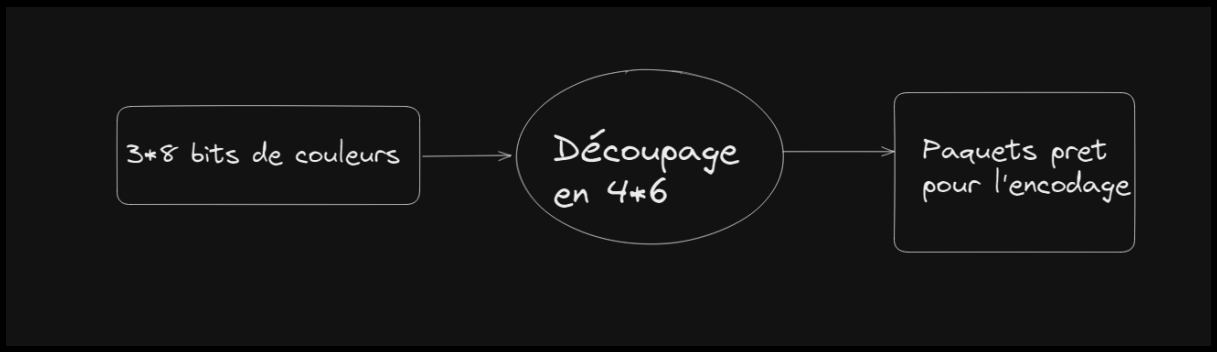
A°) Menu

Comme expliqué précédemment, le menu est un affichage interactif qui attend l'appuie d'une touche reliée à une commande. Par exemple, si l'on appuie sur la touche 1 , la commande getSize est lancée:

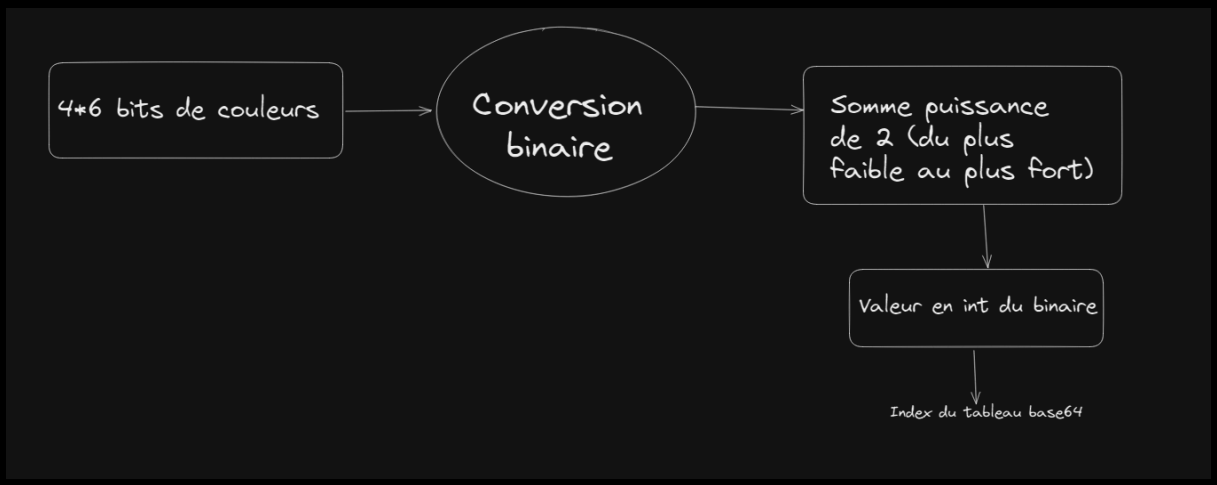


B°) Codage Base64

Pour coder en base64 la couleur, il faut d'abord quelque étape préliminaire. Premièrement, la couleur RGB est codé en 8 bits *3, alors que le base64 fonctionne sur la base de bit de taille 6. Il faut donc réorganiser les bits pour les transformer en 4*6.



Une fois les bits de longueur 6 fait, il faut maintenant les convertir en binaire pour pouvoir les comparer avec la table base64 donnée.



Ainsi , on peut obtenir la valeur en int du binaire et donc l'index du caractère en base64 équivalent à la couleur. On répète pour les 4 bits et notre couleur est maintenant en base64

6°) Conclusion

Pour conclure, nous pensons avoir répondu à la plupart des exigences du cahier des charges. Il est cependant important de dire que certaines parties ne sont pas fonctionnelles comme le getMatrix qui ne renvoie pas la totalité de la matrice. Ce projet a été pour nous le début de la découverte du monde du réseau et nous a permis de mieux comprendre la complexité de celui-ci.