Projet Bataille navale

Architecture du projet (segmentation du code)

Nous avons décidé de segmenter notre projet en package en fonction des rôles des classes et interfaces écrites : *gameStruct* pour le code des entités du jeu (navire, terrain de bataille …), *Client* pour la programmation réseau côté client et *Serveur* pour le programme côté serveur.

Choix de conception

- **Communication :** Nous avons mis sur pied un protocole de communication afin de s'assurer au mieux possible de l'intégrité des données reçues par le serveur : Le client est toujours à l'écoute du serveur et ne lui envoi des messages que s'il les lui demande. Ainsi, le serveur envoi des instructions aux clients sous ce format : *instruction : complément d'instructions.* Le serveur est en permanence à l'écoute et les traite au fur et à mesure qu'elles arrivent. Les différentes instructions envoyées sont : **WAIT** (patienter avec un message en complément), **POS** (demander l'initialisation des positions des navires), **GET** (recevoir les dégâts de l'adversaire), **SHOOT** (envoyer les coordonnées de tirs), **RESULT** (recevoir le résultat de son tir précédent), **SUNK** (navire coulé plus le nom du navire en complément) et **OVER** (partie terminée avec le résultat final). Ce protocole de communication rend le code plus lisible et offre la possibilité de faire une refonte totale d'une partie de l'application sans que l'autre ne soit affecté ! De plus, on pourrait rajouter des fonctionnalités sans avoir à tout refaire le code
- **Les erreurs** : Afin de simplifier le nombre d'exécutions côté serveur, nous n'envoyons des données au serveur que si elles sont dans un format bien défini. Nous demandons donc à l'utilisateur de saisir des données correctes s'il se trompe. Toutefois, nous limitons le nombre d'erreurs chaque fois et tirons passons son tour en tirant à l'origine
- **Données de jeu** : Quatre navires de tailles prédéfinies et positionnés sur une grille de taille 10. Ces données sont facilement modifiables, mais pas par le client. Les données du jeu sont stockées côté serveur, le client possède naturellement les informations sur son navire, et les complète à chaque tour
- Pour limiter le volume de données et le nombre de messages échangé sur le réseau, le client termine l'initialisation des position (et toutes les vérifications que cela implique) avant transmission au serveur.
- Respecter au maximum le principe SOLID (il a pas été totalement respecté)

Choix de programmation :

- Pour implémenter la communication, un interpréteur de message pour le client dans la classe ***ListenningThreadClient*** et exécution synchronisée de la communication coté serveur dans la classe ***ThreadServer***
- Aucun lancement d'exception dans la gestion des données entrées par l'utilisateur.

- La classe ***Ship*** représente un bateau. Cette dernière contient un dictionnaire pour la taille des bateaux en fonction de leurs noms et des méthodes notifiant lorsqu'il est touché, coulé, ou un problème de positionnement sur la grille.
- Une classe abstraite ***BattleField*** (dont hérite les classes ***ClientBattleField*** et ***ServerBattleField***) contient une méthode généralisée d'initialisation des champs de bataille.
- Les méthodes d'initialisation de position des bateaux, de shoot et réception de dégâts d'un client sont implémentés dans ***BatailleNavaleClient***.
- Parties intéressantes du code à regarder : ListenningThreadClient, ThreadServer, BatailleNavaleCLient, ServerBattleField.