SNAIL TECH

Génie & qualité logiciel

Projet Java

BattleShip

Clément Diot — Développeur interface
Nicolas Loisy — Ingénieur réseau
Alexis Leon — Développeur Java sénior
Ita Maknine — Consultant

Siège Social : Avenue de la Salade - 75001
Contact : +33 00 01 02 03 04 - battleship@snailtech.com - www.snailtech.com

Version 3.4.2 - 30/05/2023

Table des matières

I Introduction

Après avoir pris en compte les réflexions du groupe, nous avons décidé de développer un jeu de bataille navale "*BattleShip*" en utilisant le langage **Java**.

Nous avons commencé par concevoir la version console du jeu afin d'établir une base solide avec toute la logique du jeu. Cette première version permettait un jeu en 1 contre 1 (humain vs humain) sur la même machine. Par la suite, nous avons ajouté des fonctionnalités telles que le jeu en réseau, une IA aléatoire (humain vs ordi) et une tentative d'interface **JavaFX**. L'objectif final était d'avoir une interface permettant à chaque joueur de placer ses bateaux sur son plateau en prenant en compte leur taille respective. Cependant, en raison de contraintes de temps, nous n'avons pas pu mettre en place l'interface **JavaFX**. Nous avons préféré nous concentrer sur la propreté du livrable final.

II Lien entre la théorie et la pratique

En se basant sur les fonctionnalités que nous avons implémentées dans notre projet, nous avons formulé les besoins spécifiques des utilisateurs liés à ces fonctionnalités.

Titre : Partie de bataille navale humain vs bot en réseau local

User story 1 : En tant que joueur, je souhaite me connecter au serveur local afin de jouer à une partie de bataille navale contre un bot.
Critère d'acceptation :

- La console de l'application du client lui demande de saisir son nom de joueur.
- Une fois que l'utilisateur a saisi son nom, le client établit une connexion avec le serveur local.
- Une fois connecté au serveur, le joueur est automatiquement assigné à une partie contre un bot.
- Le bot place aléatoirement ses bateaux sur son propre plateau de jeu (géré par le serveur).
- La console utilisateur du client affiche le plateau de jeu du joueur.
- Le joueur saisit les coordonnées ainsi que la position des bateaux qu'il doit placer.
- Le client envoie les informations de placement au serveur pour validation.
- Le serveur vérifie que les positions choisies respectent les règles du jeu (par exemple, pas de chevauchement de bateaux).
- Le joueur commence en entrant les coordonnées du plateau de jeu du bot pour lancer une attaque.
- Le client envoie les informations d'attaque au serveur.
- Le serveur vérifie si l'attaque touche un bateau ou non.
- Si l'attaque touche un bateau, le serveur met à jour les informations du plateau de jeu adverse.
- La console client affiche le résultat de l'attaque (touché ou raté ou coulé).
- Le joueur peut rejouer si le résultat de l'attaque est "touché" ou "coulé", sinon c'est au tour du bot.
- Le bot effectue une attaque au hasard sur une position valide du plateau

de jeu du joueur (géré par le serveur).
— Le combat se poursuit en alternant les attaques jusqu'à ce qu'un joueur gagne en coulant tous les bateaux de son adversaire.
— La console client affiche un message de victoire pour le joueur si tous les bateaux du bot sont coulés.
— La console client affiche un message de défaite pour le joueur si tous ses propres bateaux sont coulés.

Titre : Partie de bataille navale humain vs humain en réseau local

User story 2 : En tant que joueur, je souhaite me connecter au serveur local afin de jouer à une partie de bataille navale contre un autre humain.
Critère d'acceptation :
— La console de l'application du client lui demande de saisir son nom de joueur.
— Une fois que l'utilisateur a saisi son nom, le client établit une connexion avec le serveur local.
— Une fois connecté au serveur, le serveur attend la connexion local d'un autre joueur (2 joueurs max acceptés par le serveur).
— La console client affiche un message d'attente d'un autre joueur.
— Une fois un autre joueur connecté, la console utilisateur du client affiche le plateau de jeu du joueur.
— Le joueur saisit les coordonnées ainsi que la position des bateaux qu'il doit placer.
— Le client envoie les informations de placement au serveur pour validation.
— Le serveur vérifie que les positions choisies respectent les règles du jeu (par exemple, pas de chevauchement de bateaux).
— Le serveur attend que les deux joueurs aient placé tous leurs bateaux avant de commencer la partie.
— Le joueur qui s'est connecté en premier au serveur commence en entrant les coordonnées du plateau de jeu de l'adversaire pour lancer une attaque.
— Le client envoie les informations d'attaque au serveur.
— Le serveur vérifie si l'attaque touche un bateau ou non.
— Si l'attaque touche un bateau, le serveur met à jour les informations du

plateau de jeu adverse.
— La console client affiche le résultat de l'attaque (touché ou raté ou coulé).
— Le joueur peut rejouer si le résultat de l'attaque est "touché" ou "coulé", sinon c'est au tour du joueur adverse.
— Le joueur attend l'attaque adverse.
— La console client affiche le résultat de l'attaque adverse (touché ou raté ou coulé).
— Le joueur adverse peut rejouer si le résultat de l'attaque est "touché" ou "coulé", sinon c'est au tour du joueur.
— Le combat se poursuit en alternant les attaques jusqu'à ce qu'un joueur gagne en coulant tous les bateaux de son adversaire.
— La console client affiche un message de victoire pour le joueur si tous les bateaux du bot sont coulés.
— La console client affiche un message de défaite pour le joueur si tous ses propres bateaux sont coulés.

Dans un second temps, nous avons réalisé un diagramme de classe pour la partie client de notre application :

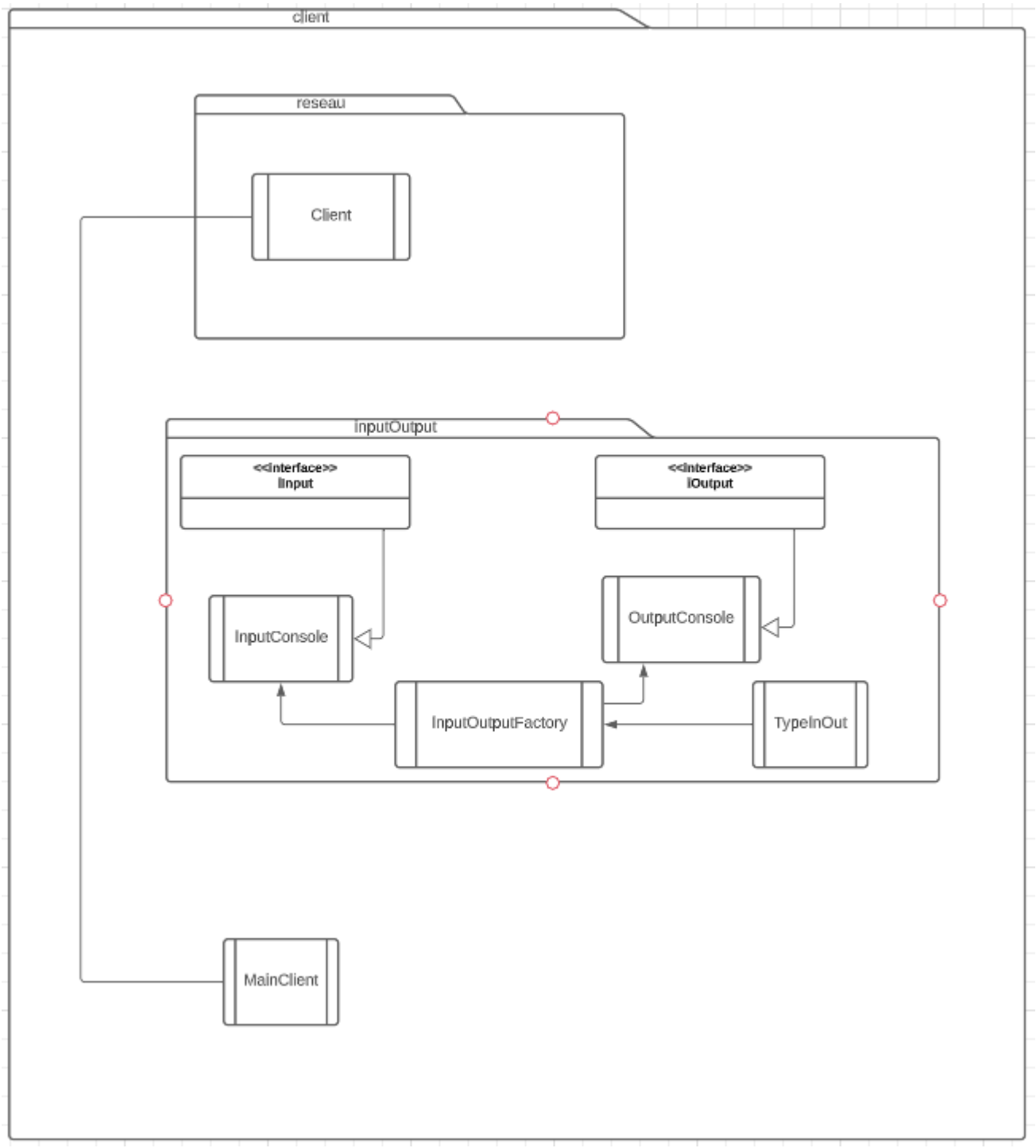

Figure 1 – Diagramme de classe de la partie client

Et nous avons également réalisé un autre diagramme de classe pour la partie serveur de notre application :

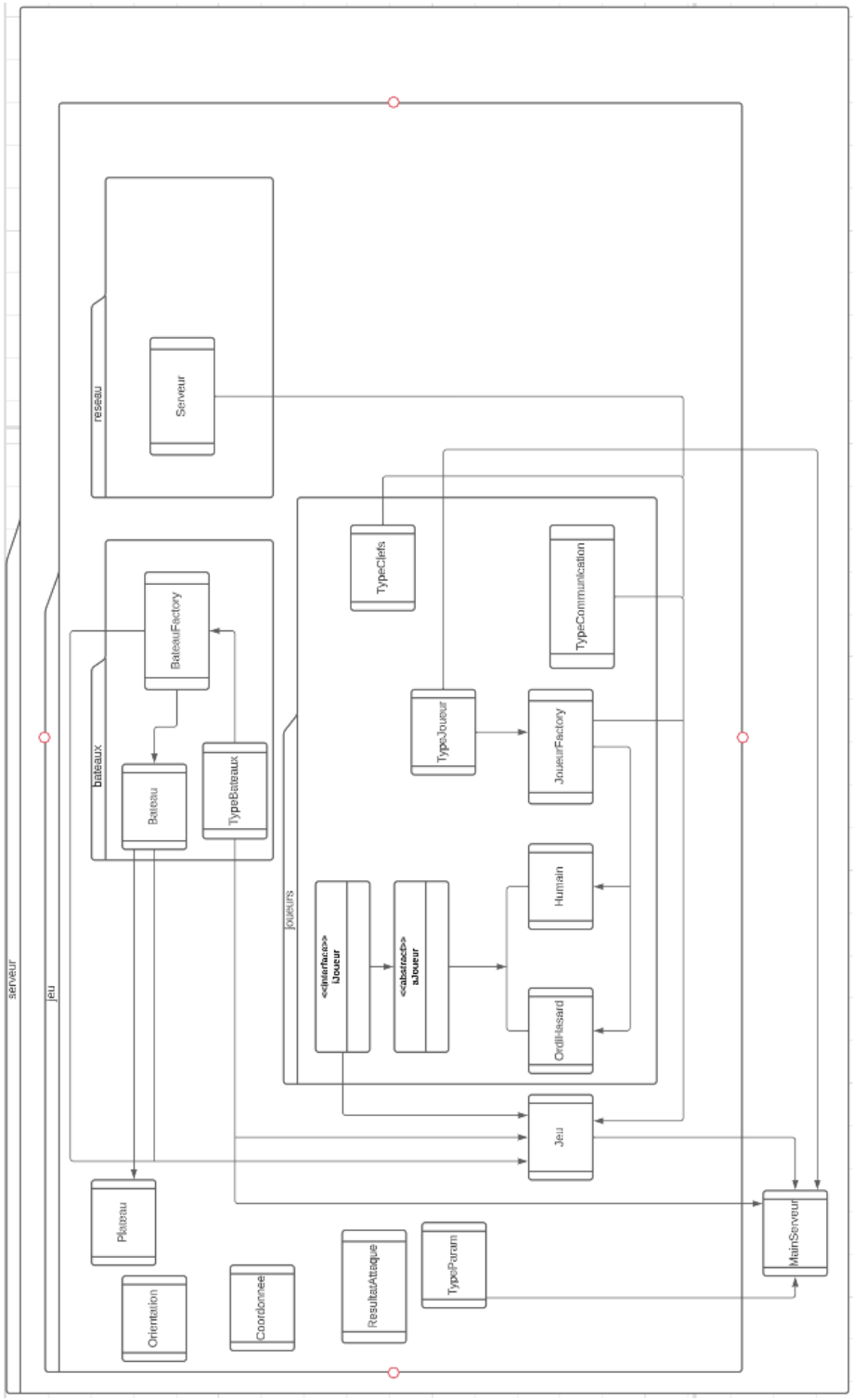

Figure 2 – Diagramme de classe de la partie serveur

III Organisation

L'organisation du projet a été centrée sur la communication et l'utilisation d'outils de collaboration. Nous avons créé un dépôt sur **Git/GitHub** pour faciliter la collaboration sur le code, gérer les différentes versions et rendre le projet accessible en ligne. De plus, nous avons mis en place un canal privé sur **Discord** dédié aux discussions autour du projet. Pour la rédaction du rapport, nous avons utilisé **Overleaf** (Overleaf est un éditeur LaTeX en ligne, collaboratif en temps réel).

Une fois les questions liées à la collaboration et à la communication résolues, nous avons réparti les tâches en fonction des besoins du projet et des envies de chacun. Le développement du jeu de bataille navale en **Java**, la communication réseaux et la conception de son interface graphique en **JavaFX** étaient les aspects principaux du projet.

Le développement du programme a suivi différentes étapes, avec un accent particulier sur la **maintenabilité** du code. Cela a inclus :

— La conception logique du programme
— L'isolation des entrées et des sorties
— La création d'un adversaire jouant au hasard pour les tests
— L'ajout de la fonctionnalité réseau
— Le refactoring complet du code
— L'ajout d'un logger
— La mise en place de tests unitaires

En ce qui concerne l'interface graphique, nous avons choisi d'utiliser **JavaFX**. Un membre du groupe avait déjà des bases sur cette technologie, ce qui nous a permis de nous former plus facilement. Nous avons conçu les principaux écrans du jeu, tels que :

— un écran de menu pour rejoindre une partie contre un ordi ou un humain.
— un écran pour placer ses bateaux.
— un écran de jeu comportant deux plateaux (l'un recevant les coups de l'adversaire sur nos navires et l'autre pour attaquer le joueur adverse).

Cependant, vers la fin du projet, nous avons décidé de mettre temporairement de côté la partie graphique afin de nous concentrer sur l'amélioration de la qualité du code et la rédaction de la synthèse du projet.

IV Difficultés rencontrées et Solutions adoptées

Code de l'application	
Problème	**Solution**
Maintenabilité du code	Revue de la conception
Adaptation en réseau	
Problème	**Solution**
Compétences limitées	Remise à niveau nécessaire à l'aide d'anciens cours
Première réflexion : avoir une machine qui héberge la partie, l'autre qui est client	Un serveur séparé des joueurs qui héberge la partie, les joueurs sont tous les deux clients
Les échanges de données n'étaient pas adaptés au réseau	Changement du système de communication
Interface JavaFX	
Problème	**Solution**
Compétences limitées	(Re)mise à niveau nécessaire à l'aide d'anciens cours, projets et documentation
Problème bloquant et chronophage sur la configuration d'eclipse pour implémenter JavaFX	Solution trouvée grâce à la documentation
Problème système de drag and drop pour placer les bateaux	Problème non résolu
Problème d'architecture de fichiers du programme principal entraînant une difficulté pour intégrer la partie JavaFX	Problème non résolu
Les changements apportés suite à la configuration du réseau ont remis en question l'idée de l'interface de départ, ce qui a donc ralenti sa création	L'intégration de l'interface a donc dû être décalée et enfin annulé
Maven	
Problème	**Solution**
Mauvaise architecture fichiers pour accueillir l'outil de gestion de dépendances	Installation de Maven, lancement du projet et compilation par command maven
L'ajout de Maven empêchait le chargement et l'accès au fichier (FR.properties)	Correction partielle de la structure pour que Maven trouve le fichier properties

Table 1 – Problèmes rencontrés et solutions trouvées

V Améliorations à apporter, tâches restantes et points non solutionnés

En plus des problèmes mentionnés précédemment, nous avons identifié certains aspects qui nécessitent encore des améliorations et des tâches restantes à accomplir. Malheureusement, en raison de contraintes de temps, nous n'avons pas pu les résoudre totalement :

— Concernant le jeu en réseau, nous avons réussi à le faire fonctionner en local, mais nous n'avons pas pu le configurer pour une utilisation avec une adresse IP publique. Une solution potentielle consisterait à ouvrir les ports nécessaires, mais cela n'a pas été testé car nous manquions de connaissances en matière de sécurité réseau.

— Nous avons actuellement une intelligence artificielle (IA) aléatoire avec laquelle les joueurs peuvent jouer, mais nous n'avons pas eu le temps d'implémenter une IA plus avancée qui aurait pu offrir une réflexion stratégique plus poussée, tant pour le placement des bateaux que pour les attaques. Cette amélioration a été reléguée en arrière-plan afin de prioriser d'autres fonctionnalités plus importantes.

— Certaines fonctionnalités de jeu prévues, telles que la possibilité de personnaliser les paramètres du jeu côté client (comme la taille du plateau de jeu ou le nombre de bateaux) et l'interface graphique JavaFX n'ont pas pu être implémentées faute de temps.

Il est important de noter que malgré ces points non résolus, nous avons mis l'accent sur la réalisation des fonctionnalités essentielles du jeu et sur la qualité générale du projet.

VI Retours personnels

Nicolas

En conclusion, ce projet de jeu de bataille navale a été une expérience enrichissante pour moi. J'ai pu mettre à profit mes connaissances en réseau Java et en sockets, ce qui m'a satisfait et a renforcé ma confiance dans ces domaines. De plus, mon niveau sur GitHub était déjà solide, ce qui m'a permis de collaborer efficacement avec mes coéquipiers et de gérer le contrôle de version de manière fluide.

Cependant, j'ai réalisé à la fin du projet que l'installation de Maven dès le début aurait été bénéfique. En l'installant tardivement, j'ai rencontré quelques difficultés qui auraient pu être évitées si j'avais intégré Maven dès le début du projet. J'ai pris conscience de l'importance de planifier et de configurer correctement les outils et les dépendances dès le début d'un projet pour faciliter le développement ultérieur.

Par ailleurs, j'ai constaté que l'implémentation de l'interface JavaFX était complexe et nécessitait une compréhension approfondie. Cela m'a donné envie d'approfondir mes connaissances dans ce domaine pour pouvoir tirer pleinement parti des fonctionnalités offertes par JavaFX dans mes futurs projets.

Dans l'ensemble, ce projet m'a permis de renforcer mes compétences en réseau Java, de consolider mon niveau sur GitHub et de prendre conscience de l'importance d'intégrer Maven dès le début du projet.

Alexis

Dans l'ensemble ce projet a été une réussite. Je suis fier de la contribution que j'ai pu apporter, en particulier pour la base logique. Cette partie de la programmation m'a permis de revoir l'ensemble des connaissances que j'ai pu acquérir en DUT.

J'ai également pu monter en compétences sur la programmation réseau en Java grâce à l'aide de mon équipe.

Ma seule frustration est l'impossibilité d'aboutir certaines fonctionnalités initialement prévues. Cela s'explique par des difficultés à maintenir la motivation au sein de l'équipe durant toute la durée du projet.

Clément

Ce projet a été une occasion pour moi de me remettre en question et d'évaluer mes compétences. J'ai décidé de relever le défi personnel de travailler sur une nouvelle technologie inconnu pour moi : le JavaFX. Cependant, en raison de mon obstination et de ma réticence à demander davantage d'aide à mon groupe sur cette partie, le développement de l'interface JavaFX n'a pas été inclus dans notre application finale. C'est une frustration personnelle et pour l'ensemble du groupe. Néanmoins, je tiens à exprimer ma gratitude envers mes collaborateurs qui ont cru en moi et qui ont, même sans ma partie graphique, proposé un code maintenable, propre et fonctionnel.

Cette expérience m'a cependant permis d'acquérir des connaissances ainsi qu'une base en JavaFX, que je pourrai réutiliser dans de futurs projets. Enfin, ce projet m'a également permis de voir l'organisation d'un projet à plus long terme. Et concernant les cours, j'ai énormément appris et aimé comment les cours étaient présentés avec l'ajout de nombreuses références accompagnées de leurs exemples.