Compte rendu de TP — MEDEVRV — Bataille Navale

Cheikh Oumar Ba, Narjisse El Massouri, Florian Barbe
Option Réalité Virtuelle

1 Introduction

Ce travail pratique a pour objectif la conception et l'implémentation d'un jeu de bataille navale en C++, dans lequel un joueur affronte un ordinateur. Le TP vise à mettre en place une architecture logicielle cohérente, à produire une modélisation UML (diagramme de classes et diagramme d'état-transition), puis à développer un programme fonctionnel.

Le jeu se déroule sur deux grilles de taille 10×10 : celle du joueur et celle de l'ordinateur. Chaque participant possède une flotte composée de cinq bateaux de tailles différentes. Le joueur place manuellement ses bateaux, tandis que l'ordinateur les place aléatoirement. Les deux adversaires jouent ensuite à tour de rôle en tentant de toucher puis couler l'ensemble de la flotte ennemie.

2 Objectifs du TP

Les objectifs principaux étaient les suivants :

- modéliser la structure du logiciel (diagramme de classes UML et diagramme d'état-transition),
- définir une stratégie de développement et une répartition des tâches,
- implémenter une version fonctionnelle du jeu en C++,
- produire une documentation utilisateur et une documentation technique,
- mettre en place un dépôt Git contenant l'ensemble du projet.

3 Conception du logiciel

Pour structurer le programme, nous avons mis en place une architecture modulaire reposant sur plusieurs classes : `grille`, `bateau`, `ChoixPlacement`, `Tir`, ainsi qu'un module `utilities` contenant l'ensemble des fonctions opérationnelles (tir, validation, mise à jour, etc.).

3.1 Diagramme de classes UML

Le diagramme suivant présente les relations entre les différentes classes du projet ainsi que leurs attributs et méthodes principaux.

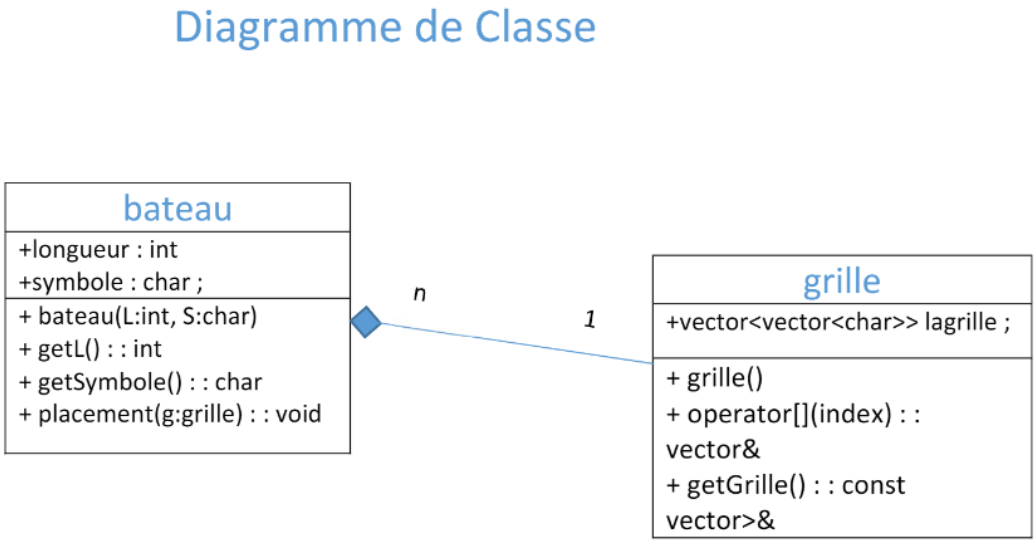

FIGURE 1 – Diagramme de classes UML

3.2 Diagramme d'état-transition

Le déroulement complet d'une partie est modélisé par un diagramme représentant les états successifs : placement des bateaux, boucle de jeu (tour joueur / tour ordinateur), et conditions de victoire/défaite.

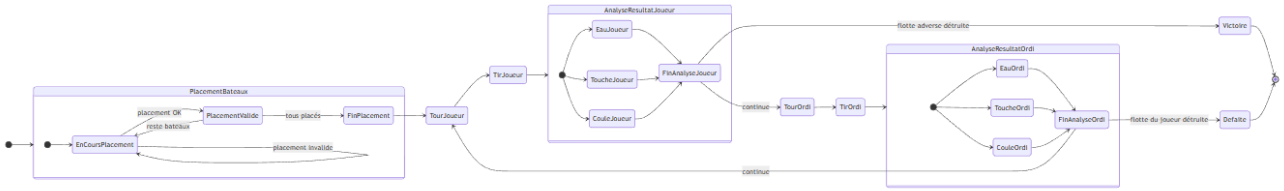

FIGURE 2 – Diagramme d'état

4 Développement du programme

4.1 Structure générale du code

Le code est organisé en plusieurs fichiers :

— **grille.h / grille.cpp** : gère la matrice 10×10, l'affichage et l'accès aux cases,

— **bateau.h / bateau.cpp** : stocke longueur et symbole du bateau et gère son placement aléatoire,

— **ChoixPlacement.h** : lit et valide les entrées utilisateur,

— **Tir.h** : structure contenant des coordonnées (x, y),
— **utilities.h / utilities.cpp** : contient toutes les opérations essentielles :
  — `placement_valide` : vérifie que le bateau tient et ne chevauche pas,
  — `placer_bateau` : inscrit les symboles sur la grille,
  — `lire_tir_joueur` : saisie utilisateur pour une coordonnée de tir,
  — `tir_ordi` : sélection aléatoire d'une case valide,
  — `evaluer_tir` : renvoie le résultat (*à l'eau*, *touché*, *coulé*).
— **main.cpp** : enchaîne placement, boucle de jeu, affichage et conditions de fin.

4.2 Placement des bateaux

Le joueur place ses bateaux en choisissant :
— le type (`P`, `CR`, `CT`, `S`, `T`),
— les coordonnées (x, y),
— l'orientation (`H` ou `V`).

L'ordinateur, lui, place ses bateaux automatiquement à l'aide d'une fonction de placement aléatoire contrôlée.

4.3 Boucle de jeu

Le joueur et l'ordinateur jouent chacun leur tour. Chaque tir modifie la grille :
— `O` : tir raté,
— `X` : tir touchant un bateau.

Si toutes les cases d'un bateau sont touchées, la fonction `evaluer_tir` renvoie l'état `COULE`. Le joueur gagne lorsque tous les bateaux ennemis sont détruits, et inversement.

5 Répartition des tâches

Nous avons commencé par réfléchir à la logique du jeu tous ensemble. Le principal débat a porté sur le nombre de grille à définir. Nous avons finalement décidé de jouer avec deux grilles, une pour l'ordinateur et une pour l'humain. Au départ nous avions également prévu de faire des classes filles pour chaque type de bateau. Nous les avions codé mais finalement cela a posé problème sur la mécanique du jeu, le rendant complexe alors qu'il n'y avait pas de réel intérêt à les différencier puisqu'ils ont tous le même comportement. Nous avons donc parlé entre nous au fur et à mesure pour adapter chaque partie du code.

Au départ nous travaillons tous en même temps et parfois sur le même fichier. Nous avons rencontré beaucoup de difficultés au moment de merge. Les conflits s'accumulaient et nous devions push et pull à chaque ligne pour avoir la même version. Finalement nous avons réussi à trouver une configuration plus confortable. On a recréer une solution et un fichier .cpp et .h par personne. Chacun ne modifiais que son code.

A la fin, Florian en faisant une relecture de code est repassé sur l'ensemble des fichier pour que le programme puisse compiler. Il a adapté le nom des fonctions si nécessaire et rajouté les attributs qu'il avait utilisé dans ces fonctions s'ils n'étaient pas déjà présent. Avec du recul, nous aurions dû réfléchir plus longtemps ensemble sur le nom des classes et attributs utilisés.

Aa tâche	responsable	estimation du temps (he...	État
Classe grille et bateau	Cheikhou	4	Terminé
placement du bateau	Narjisse	1	Terminé
mécanique du jeu avec tour humain et tour du robot (main)	Narjisse	4	Terminé
classe fille des bateaux	Cheikhou	0	annulé
fonctions du main (tirer, touché ...)	florian	5	Terminé
rédaction du rapport	Cheikhou	1	En cours

FIGURE 3 – Répartition des tâches

6 Métrique de SourceMonitor

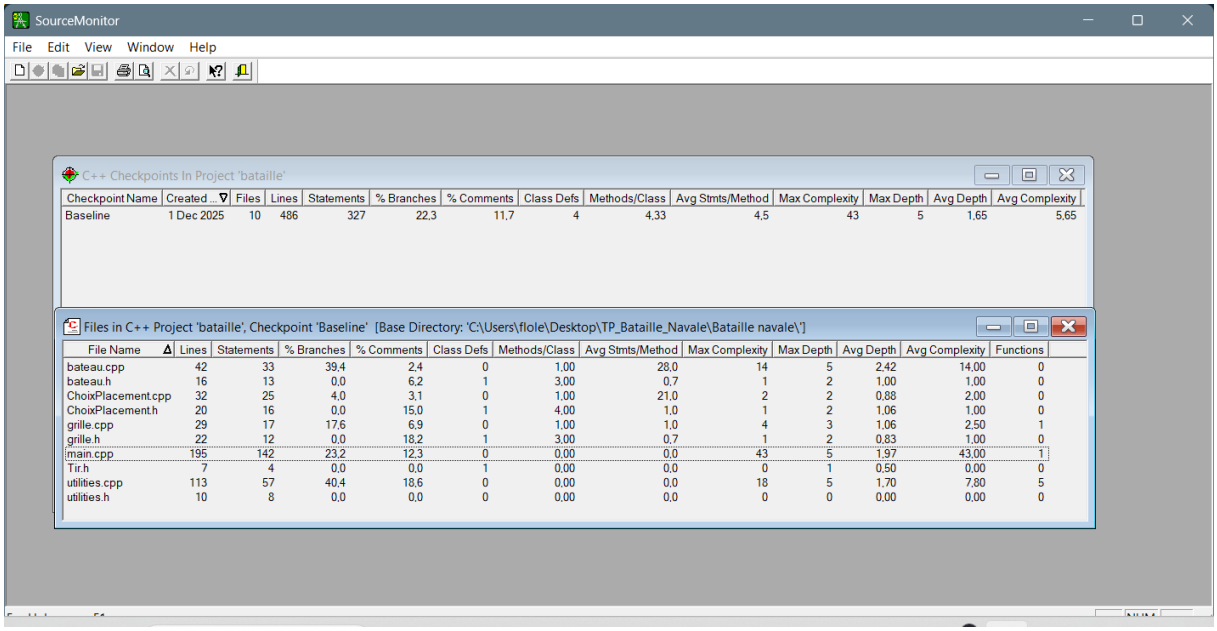

FIGURE 4 – Mesures des complexités du code avec SourceMonitor

Le projet comporte environ 450 lignes de code réparties sur une dizaine de fichiers, ce qui correspond à une taille modeste. La complexité cyclomatique moyenne est correcte (5), montrant une logique globalement simple et appropriée. Cependant, la complexité maximale atteint 43 dans main.cpp, ce qui révèle une fonction très longue, difficile à maintenir et à tester.

La profondeur d'imbrication atteint 7 à 8 niveaux dans certains cas, réduisant la lisibilité et rendant le flux d'exécution plus difficile à suivre. Une restructuration (extraction de sous-fonctions, séparation des responsabilités) serait bénéfique.

La répartition du code montre que la majorité de la logique du jeu est centralisée dans main.cpp, alors que les autres fichiers restent peu complexes. Une architecture plus modulaire améliorerait la maintenabilité et l'évolutivité du projet.

7 Documentation utilisateur

L'utilisateur :

1. lance le programme depuis un exécutable,
2. place ses cinq bateaux via des entrées clavier en suivant les indications données dans la console,
3. observe sa grille et celle visible pour l'adversaire,
4. effectue un tir par tour en indiquant des coordonnées,
5. remporte ou perd la partie selon le résultat de la boucle de jeu.