Programmation objet – Langage C++

Projet Smoke : BattleShip

Walid Ghettas Marc Bassecourt

POLYTECH SORBONNE

Dans le cadre de l'UE Langage Objet C++, nous avons eu comme thème de projet le mot "Smoke" (fumée). Afin de couvrir le thème donné et correspondre aux contraintes imposées, nous avons choisi de réaliser une bataille navale, où la fumée couvre alors l'écran du joueur lorsqu'il tire.

Les règles du jeu sont simples :
-> Chaque joueur dispose de 4 bâtiments à placer :

- Longship de 5 cases
- Middleship de 4 cases
- Shortship de 3 cases
- Phare de 2 cases

-> Les joueurs jouent tour à tour en donnant la position à bombarder, s'il trouve une position ennemie, alors c'est touché, s'il détruit entièrement un bâtiment ennemi, alors c'est coulé. En revanche, si la position visée est inoccupée, c'est raté!

-> Si le phare est touché, il éclaire dans une zone de 3 cases autour de lui, il faut donc placer stratégiquement votre phare pour qu'il soit à la fois dur à trouver et loin de vos bateaux.

-> Lorsque tous les bâtiments d'un joueur sont détruits, alors c'est gagné!

Par soucis de complexité, nous avons choisi de développer le jeu sur une seule et unique console plutôt qu'en réseau, sachant de plus que le jeu se jouant à 2 personnes, l'utilisation d'une seule machine n'est pas contraignante.

La structure de notre Bataille Navale

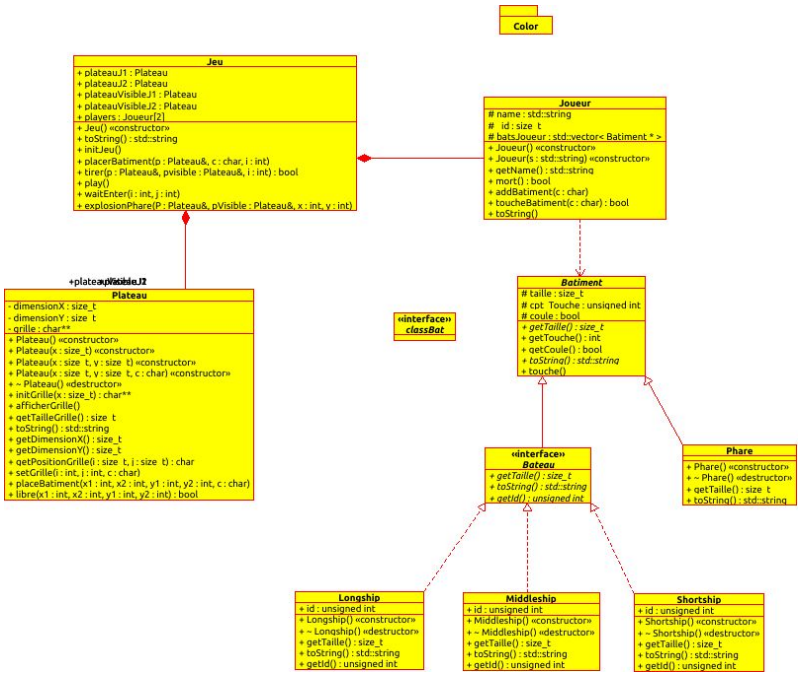

Diagramme UML de notre projet de Bataille Navale

Comme le montre l'UML ci-dessus, notre projet respecte plusieurs contraintes parmis lesquelles : plus de 8 classes, 3 niveaux de hiérarchie, plus de 2 fonctions virtuelles (classes abstraites), utilisation de containers.

Notre jeu repose alors sur des grilles fixes appelées Plateaux (donc des tableaux), contenant des caractères représentatifs (L,M,S,P pour les bâtiments et 0 pour l'eau). Il y a ensuite des joueurs, possédant une liste de bâtiments ainsi qu'un nom et diverses fonctions que l'on développera par la suite.

Le tout est alors rassemblé dans une classe Jeu, possédant :

- deux plateaux (un pour chaque joueur) contenant la position des bâtiments.
- deux plateaux (un pour chaque joueur) correspondant au plateau visible par chaque joueur, et donc rempli de fumée au départ.
- deux Joueurs (possédant chacun leur liste de bâtiments)

Comment lancer une partie ?

Afin de pouvoir jouer, il suffit de récupérer le dossier comprenant le code source et le compiler en effectuant un "make" dans le terminal. Pour lancer une partie, il faut alors écrire la commande "./jeu".

La partie commence alors, suivez les instructions!

Le mécanisme du code

Tout le jeu repose sur une fonction unificatrice nommée Jeu::play. En effet, celle-ci appelle les autres fonctions nécessaires à la création et au déroulement de la partie.

1) Initialisation de la partie

Afin de lancer la partie, il faut tout d'abord initialiser cette-dernière. Tout commence alors avec l'instantiation de la classe Jeu qui a son tour instancie deux Joueurs dont les noms doivent alors être spécifiés. La fonction Jeu::initJeu permet alors à chaque joueur de placer ses bâtiments sur son plateau l'un après l'autre et ce grâce à une fonction tierce nommée Jeu::placerBatiment qui indique la taille du bâtiment à placer et attend des entrées utilisateur correspondant aux coordonnées de début et de fin du bâtiment à placer.

Pour aider les joueurs, des couleurs permettent d'identifier les axes.
Une fois les deux plateaux initialisés, les deux joueurs peuvent alors commencer à jouer en tirant tour par tour.

2) Tirs

Une fois la partie lancée, chaque joueur tir tour à tour. Pour cela, la fonction play appelle la fonction Jeu::tirer. Cette dernière permet à un joueur de tirer sur une zone, avec une saisie de la part du joueur pour spécifier les coordonnées choisies. Une fois ces dernières entrées, il y a tout d'abord vérification de la validité des coordonnées (pas de sortie de la grille) et ensuite le tir a lieu. Si la zone visée est une position ennemie, alors c'est touché! Le tireur est alors informé et sa carte s'actualise alors pour lui afficher un 'X' synonyme de succès. Au contraire, s'il ne touche pas de bâtiment ennemi, c'est raté, un 'O' s'affiche alors sur sa carte.
Après chaque tir, la fonction tirer renvoie alors un booléen, false si la partie n'est pas finie, true si elle l'est.

3) Fin de partie

Comme dit précédemment, à chaque appel de la fonction tirer, on vérifie si la partie est terminée. Pour cela, on appelle la fonction Player::mort à la fin de la fonction tirer. Cette fonction vérifie alors grâce au vecteur de bâtiments si ce dernier est vide ou non, si c'est le cas, alors le joueur est mort et la partie se termine.

Les principales technicités du projet

Une des principales difficultés du projet fut les segmentation fault assez fréquentes lors de la manipulation des multiples tableaux. En effet, plusieurs fonctions telle que Jeu::explosionPhare nécessitent de certaines précautions afin d'éviter les sorties de plateau.

Pareillement, la fonction Jeu:placerBatiment est particulièrement compliquée. En effet, il a d'abord fallu réfléchir à la manière dont on allait demander les coordonnées au joueur. Nous avons alors choisi de demander les coordonnées de début et de fin du bâtiment. Il a alors fallu trouver un moyen de valider la cohérence des coordonnées entrées. En effet, seuls les placements verticaux et horizontaux sont autorisés, la zone désignée doit être libre (d'où la fonction Plateau::libre).