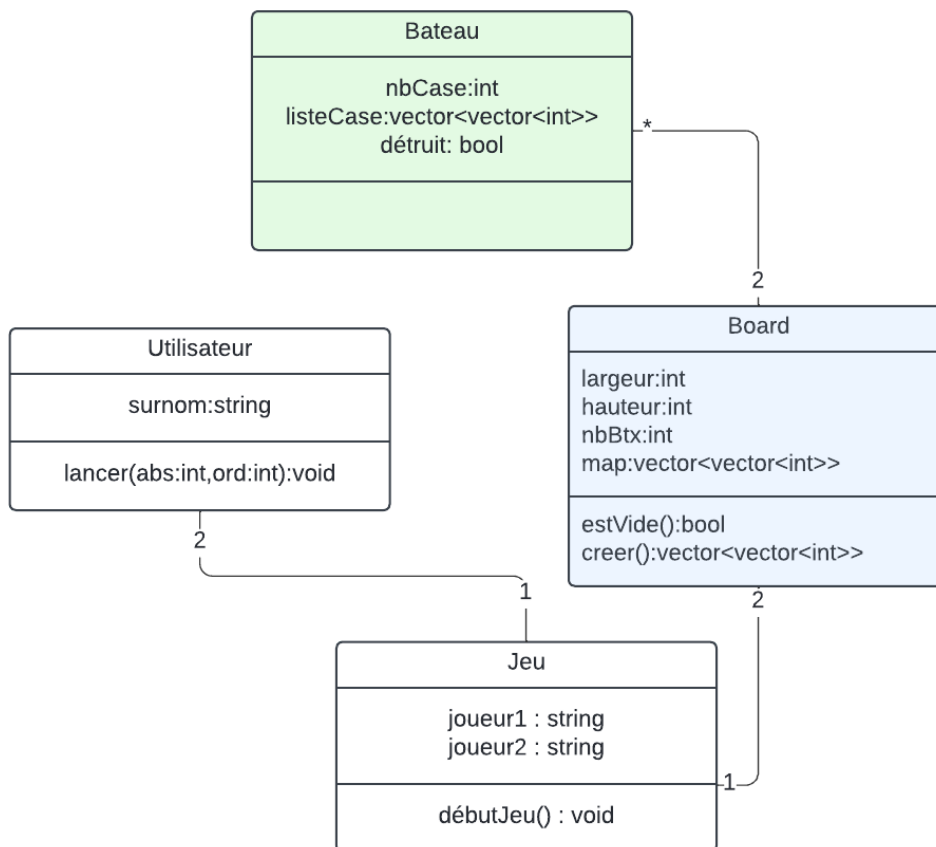


Elio NICOLE
Lory SOUMPHONPHAKDY
Bastien THIRION
Agathe MIGNOT

TP MEDEV Bataille Navale

On a commencé par faire un UML initial :



Le diagramme est plutôt simple et on se doutait que notre code finirait par changer mais on ne voyait pas quoi mettre spécialement de plus tant que nous n'avions pas commencé à coder.

Tout d'abord, on s'est dit que la classe jeu n'était pas utile et qu'on pouvait le faire dans le main plutôt.

Maintenant, en voyant la métrique de notre jeu et donc que la complexité du main.cpp est à 49, on se dit qu'il faudrait peut-être refaire une classe jeu et répartir ce qu'on a fait dans le main dans des méthodes qui seront dans la classe jeu où même dans les autres classes éventuellement.

La classe utilisateur aussi était inutile finalement, on a remplacé ça par le fait de demander le nom des utilisateurs au début et ensuite chacun joue à son tour.

On a eu des problèmes quand ce qui était rentré, pour la position et/ou pour le vertical ou horizontal, n'était pas dans l'intervalle qu'on souhaitait, cela crashait au début. On a donc géré les autres cas quand on a eu le temps.

On voulait pouvoir entrer des lettres pour correspondre au jeu de la bataille navale normale qui a des coordonnées du type lettre nombre. C'est pourquoi on a dû voir les nombres ascii correspondant aux lettres de A à J.

On a une erreur que nous n'avons pas eu le temps de régler : si quelqu'un rentre d'abord en coordonnées un chiffre puis une lettre cela fait une boucle infinie et nous n'avons plus la possibilité de rentrer les coordonnées.

Beaucoup de problèmes ont été réglés vers la fin, c'est pourquoi le code pour l'IA n'inclut pas toutes les modifications faites pour le jeu à 2.

Le jeu permet de jouer soit contre quelqu'un d'autre, soit contre une IA.

On a fait en sorte qu'une fois que le premier joueur a fini de placer ses bateaux, le terminal se clear pour que cela ne spoil pas le joueur 2. Quand on joue au jeu, on affiche la grille de l'adversaire et la nôtre, on peut voir sur la grille de l'adversaire où on a tiré (et si on a touché). Sur la nôtre, on voit nos bateaux et où l'ennemi a tiré.

Pour le jeu contre l'IA, on a créé une classe IA puis on a fait une fonction qui prend en paramètre une grille (ce sera la grille où l'IA tire).

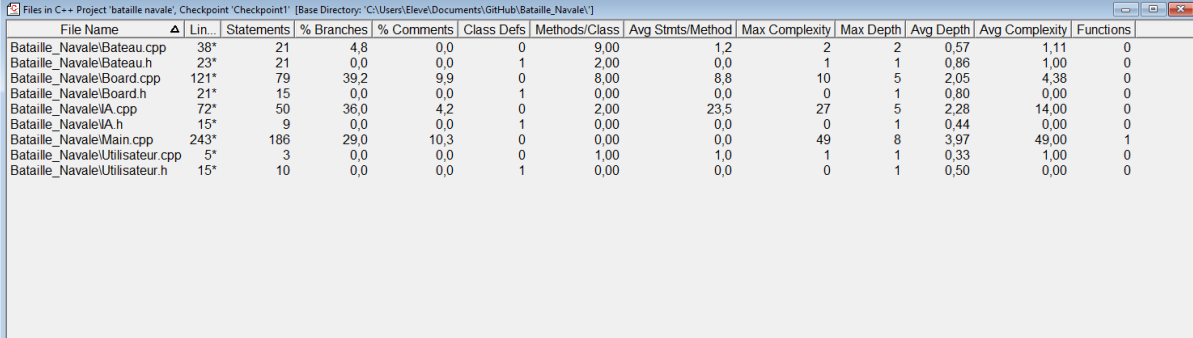
On a une première boucle if qui vérifie si deux coups ont déjà été joués, et si les deux derniers coups étaient soit sur la même ligne soit sur la même colonne. La seconde boucle regarde si les coups étaient sur une colonne ou sur une ligne. Si c'est sur une ligne, on regarde s'il y a déjà eu d'autres coups sur cette ligne, qui collent les deux derniers coups, pour savoir où tirer. En fonction de si il y a eu un tir qui n'a pas touché de bateau ou pas on tire soit à gauche soit à droite, en restant sur la même ligne. On doit toujours vérifier si le tir peut être joué (si on reste dans la grille et si la case a déjà été touchée). On a une même boucle s'il s'agit d'une colonne.

Si il y a au moins un coup et que le dernier a touché un bateau, il faut prendre au hasard une case à côté de celle-ci, en vérifiant qu'on est bien sur la grille et que la case n'a pas déjà été tirée dessus.

Dans le cas échéant, on prend une case au hasard dans la grille, avec toujours les mêmes contraintes.

Le random que nous utilisons n'est pas vraiment aléatoire car on n'utilise pas le temps donc à chaque fois l'IA joue la même chose au début. Nous ne regardons pas la taille des bateaux coulés, si nous avons plus de temps nous aurions ajuster le code en conséquence. L'IA ne prend pas en compte quand un bateau est coulé, donc si le bateau vient d'être coulé après les deux derniers coups joués, l'IA continue de tirer dans la continuité du bateau (s'il y a de la place sur la ligne ou la colonne, selon les contraintes). De plus, nous regardons seulement les deux derniers tirs : il est possible que deux cases côte à côte ont touché un bateau, mais pas par les deux derniers tirs, donc l'IA continuera de jouer aléatoirement.

La métrique :



| File Name | Lin... | Statements | % Branches | % Comments | Class Defs | Methods/Class | Avg Stmt/Method | Max Complexity | Max Depth | Avg Depth | Avg Complexity | Functions |
|---------------------------------|--------|------------|------------|------------|------------|---------------|-----------------|----------------|-----------|-----------|----------------|-----------|
| Bataille_Navale/Bateau.cpp | 38* | 21 | 4,8 | 0,0 | 0 | 9,00 | 1,2 | 2 | 2 | 0,57 | 1,11 | 0 |
| Bataille_Navale/Bateau.h | 23* | 21 | 0,0 | 0,0 | 1 | 2,00 | 0,0 | 1 | 1 | 0,86 | 1,00 | 0 |
| Bataille_Navale/Board.cpp | 121* | 79 | 39,2 | 9,9 | 0 | 8,00 | 8,8 | 10 | 5 | 2,05 | 4,38 | 0 |
| Bataille_Navale/Board.h | 21* | 15 | 0,0 | 0,0 | 1 | 0,00 | 0,0 | 0 | 1 | 0,80 | 0,00 | 0 |
| Bataille_Navale/IA.cpp | 72* | 50 | 36,0 | 4,2 | 0 | 2,00 | 23,5 | 27 | 5 | 2,28 | 14,00 | 0 |
| Bataille_Navale/IA.h | 15* | 9 | 0,0 | 0,0 | 1 | 0,00 | 0,0 | 0 | 1 | 0,44 | 0,00 | 0 |
| Bataille_Navale/Main.cpp | 243* | 186 | 29,0 | 10,3 | 0 | 0,00 | 0,0 | 49 | 8 | 3,97 | 49,00 | 1 |
| Bataille_Navale/Utilisateur.cpp | 5* | 3 | 0,0 | 0,0 | 0 | 1,00 | 1,0 | 1 | 1 | 0,33 | 1,00 | 0 |
| Bataille_Navale/Utilisateur.h | 15* | 10 | 0,0 | 0,0 | 1 | 0,00 | 0,0 | 0 | 1 | 0,50 | 0,00 | 0 |

La complexité de 49 du main peut être améliorée comme dit précédemment. En effet, on a codé la partie de 2 joueurs et la partie de l'IA dans le main et donc on a réécrit essentiellement 2 fois les mêmes principes, ce qui n'est pas incroyable.

Celle de l'IA pourrait peut-être aussi l'être si on utilisait les fonctions `toucherPos`, etc de la classe `Board`.