Bataille Navale by Arthur Mougin

Règles

Le jeu se joue à deux. On commence par choisir la taille de la carte, (traditionnellement de 10 par 10 cases). Ensuite, chaque joueur place ses bateaux (au moins 1) à l'endroit qu'il veut sur sa carte.
Le jeu peut enfin commencer : Chacun son tour, un joueur choisit une case où faire feu. S'il touche ou coule un bateau, il peut retirer dans le même tour.
Lorsqu'un des joueurs n'a plus de vie, l'autre a gagné, on peut alors recommencer une nouvelle partie ou arrêter de jouer.

Classes

Ce jeu se base sur 3 classes : Bateau, Carte et GameMaster.
Le GameMaster contient deux Carte, et chaque Carte contient autant de Bateau que le joueur demande.

1. Bateau

La classe Bateau contient les propriétés de base de ses derniers, leurs getter Setters, un constructeur vide, un constructeur paramétré et une fonction ToString().

Propriétés

les propriétés sont :

- La taille du bateau. Elle est limitée entre 2 et 5.
- La vie qui est égale à la taille lors de la création du bateau.
- L'axe x et y pour désigner le point supérieur gauche.
- Un booléen pour savoir s'il est horizontal ou vertical.

La fonction ToString a principalement servi durant le développement des autre classes.

2. Carte

La Carte contient une Matrice de caractères ainsi qu'une liste de bateau. En plus de cela, elle contient leurs getters/setters, deux constructeurs, et plusieurs fonctions membres.

Les caractères de la matrice suivent un code précis : '-' pour une case vide, 'H' pour un bateau en bon état, 'v' pour un tir raté, 'X' pour un bateau touché, et 'O' pour un bateau coulé.

Fonctions membres

Les fonctions membres sont :

- **AddBateau** : elle prend en paramètre un bateau et retourne un booléen indiquant si elle est parvenue à l'ajouter ou pas.

- **ToString** : cette fonction prend en paramètre un booléen indiquant si elle est la carte joueur ou ennemie au moment du tour. Elle retourne une chaîne de caractère formatée contenant le titre de la carte, les repères, la matrice et le nombre de bateaux.

- **GetLine** : Cette fonction découpe le retour de **ToString** pour ne retourner qu'une ligne à la demande.

- **GetDoc** : Elle retourne une chaîne de caractère contenant les explications des caractères de la matrice. Elle est présente ici pour garder toute informations sur l'encodage de la matrice sur un seul fichier. C'est surtout un choix de maintenabilité.

- **Reset** : Elle permet d'effacer la carte si l'utilisateur n'est pas satisfait du placement de ses bateaux.

- **GetBateauByCoords** : Lorsqu'une case est tirée et qu'elle contient un bateau sain, on utilise cette fonction pour savoir quel bateau était à cette case et va perdre une vie. Si le bateau n'a plus de vie, on peut aussi savoir qu'il est coulé.

- **EstFinie** : Elle regarde s'il reste des bateaux avec des points de vie. Si c'est le cas, elle retourne faux,sinon vrai.

- **BateauCoulé** : Cette fonction s'occupe de modifier la matrice en prenant en paramètre un bateau qui vient d'être coulé. Elle remplace toutes les cases occupées par celui ci par des 'O'.

- **Tirer** : elle prend en paramètre des coordonnées et va utiliser les fonctions **GetBateauByCoords** et **BateauCoulé** au besoin. Elle retourne un entier qui indique en fonction de sa valeur ce qui s'est passé. (0 pour un plouf, 1 pour un tir déjà effectué, 2 pour un touché, 3 pour un coulé) Le traitement logique se fait en fonction du contenu d'une case et en faisant des comparaisons de caractères.

3. GameMaster

La classe GameMaster est celle qui dirige le jeu avec sa fonction Jeu. Au niveau des propriétés, elle contient deux Cartes (une par joueur) et deux incréments : un pour suivre les tours, un pour suivre l'état du jeu (0 : setup, 1 : jeu, 2:fin).

Affichage et logique secondaire

Certaines fonctions membres servent principalement à l'affichage :

- **DrawMaps** : elle utilise la fonction **GetDoc** et **GetLine** de Carte pour retourner une chaine de caractère formatée contenant les deux cartes à l'Horizontal ainsi que les informations facilitant la lecture de la Matrice.
- **PrintInputs** : Cette fonction retourne les différents inputs possibles dans un contexte particulier entré en paramètre.
- **AskUser** : C'est la fonction la plus référencée, elle prend en paramètre un message à afficher et retourne ce que l'utilisateur a retourné.
  Le contenu retourné peut être utilisé par la suite pour un formulaire ou non pour un message attendant une input utilisateur pour passer à la suite.
- **ResetView** : Elle efface la fenêtre et exécute **DrawMaps**.
- **JeuCommence** : Cette fonction membre se base sur **AskUser** pour annoncer le début du jeu
- **Victoire** : On utilise **AskUser** pour annoncer la victoire et le gagnant.
- **NextTurn** : Elle permet de laisser les joueurs changer de place en effaçant la fenêtre et en utilisant **AskUser** pour demander au prochain joueur de prendre place. Elle incrémente aussi le Tour.

D'autres fonctions sont dédiés logiques. Parmis elles, on a des Getter customs : ils ne font pas que retourner une information, il font du traitement logique avant cela.

- **GetPlayerMap** : permet de retourner une carte en fonction du tour, c'est celle du joueur.
- **GetEnemyMap** : retourne l'autre map.

On retrouve une fonction logique retournant un booléen pour traiter les inputs de coordonnées (**TestCoords**), elle est utilisée pour éviter le code dupliqué sur cette partie logique.

Logique primaire

Viens maintenant les fonctions coeur du jeu : Setup, Tirer et Jeu.

Setup

Cette classe intègre toute l'application pour le processus d'ajout d'un bateau pour un joueur. La première étape est de demander les coordonnées du bateau en utilisant **AskUser** et **TestCoords**. La seconde est demander la longueur du bateau et la troisième son orientation.

Une fois cela fait, on utilise **Carte.AddBateau** pour savoir si le bateau à put être créé. Si ce n'est pas le cas, on laisse le choix à l'utilisateur de réessayer ou non.

Tirer

Quand a elle, prend en compte la logique de tir, on retrouve le même principe que Setup pour les coordonnées. En fonction de ce que **GetEnemyMap** et **Carte.Tirer** retourne, on peut savoir si le joueur peut retirer ou non avant de passer au tir suivant.

Jeu

La fonction Jeu est la fonction qui les appellent toutent. Elle est composée comme une poupée russe de boucles Do-While.
La première boucle Est celle de jeu, elle ne s'arrête que quand on quitte le jeu.
Elle contient les différentes boucles-étapes du jeu :

- La première est le choix de taille, on demande à l'utilisateur les dimensions des matrices. On en sort quand les dimensions sont prêtes.

- La seconde étape est la boucle de Setup, on passe d'un utilisateur à l'autre grâce à **NextTurn.** Pour chaque Joueur, on le laisse choisir d'ajouter un bateau avec **Setup**, d'effacer les bateaux qu'il ne veut plus avec Carte.**Reset** ou alors de valider ses modifications.

On exécute **JeuCommence.**

- La troisième Boucle est certainement celle qui va prend le plus de temps : Elle gère le jeu et l'alternance d'un joueur à l'autre. On n'en sort que lorsqu'une des deux cartes **EstFinie**. Durant chaque tours, Le joueur peut **Tirer** sur le plateau ennemie ou passer son tour.

On exécute **Victoire .**

- On termine par demander à l'utilisateur s'il veut recommencer la boucle de jeu et revenir à la première étape ou sortie de la Boucle.

Points d'amélioration

Le design de l'interface

Pour toute applications en général, on cherche à garder des éléments visibles pour l'utilisateur et en faire disparaître d'autre. C'est encore plus technique pour une application textuelle.
Ici j'ai choisi d'utiliser souvent la fonction **ResetView** pour garder à vue la carte. Cette fonction affiche les deux cartes à l'horizontal grace à de la concaténation et l'usage de **GetLine**.

On peut encore améliorer ce concept car **GetLine** est assez peu efficace. Chaque appel régénère l'interface avec **ToString**, mais l'on en retourne qu'une seule ligne.
Une solution serait de n'appeller qu'une fois **ToString** par carte dans **ResetView** puis de faire une fonction **GetLine** qui prend une copie du retour de **ToString** en paramètre.

Le design des interactions

Jouer avec **ResetView** est difficile car c'est difficile de savoir quand garder une information à l'écran et quand effacer la fenêtre et repartir à 0.
En effet, les seuls éléments qui sont conservés sont le contenu de **ResetView.** C'est pour cela que ma gestion des erreurs d'input a quelques inconsistances quand à la réinitialisation de la fenêtre.

Pour réduire ces inconsistances, il aurait fallu regrouper toutes les demandes d'input dans une fonction qui prendrait en paramètre Les options/type d'inputs demandées (ce que j'ai commencé avec **PrintInputs**) et un tableau de fonctions à activer pour chaque input.
Cette fonction traiterait alors de manière uniforme les erreurs d'inputs.

Le design des Classes

La classe Carte est construite entièrement sur le concept de matrice. La matrice y est le saint Graal de l'information, ce qui est une erreur. En effet, si la Classe Bateau était davantage utilisée et que la matrice était utilisée uniquement à des fins d'affichage, j'aurais pu créer des bateaux plus interactifs et dynamiques (ajouter l'option de se déplacer par exemple). Et ainsi creuser davantage avec les héritages.

Conclusion

C'est un projet intéressant qui m'a fait me confronter à des manipulations de matrice et du multijoueur, ce que j'ai bien aimé.
Si c'était à refaire, je referais certainement autrement ma classe Carte, Le reste est assez solide.