TP n°4
Pirate des Caraïbes

MOdélisation OBjet (MOOB)

2024-2025

L'objectif des prochaines séances de TP est d'implanter un jeu de bataille navale.

Dans un premier temps, vous allez devoir réfléchir à la structure de votre code selon les spécifications données. Pour cela, vous allez réaliser un diagramme de classe. Une fois votre diagramme de classe validé, vous allez pouvoir commencer à implanter les fonctionnalités en suivant l'ordre proposé par le TP.

Vous avez maintenant pratiqué les bases de Java : constructeur, méthodes, héritage. Vous allez mettre en application tout cela dans un code un peu plus conséquent. A chaque étape, vous **devrez** réfléchir à la structure de votre programme et soigner l'implantation des fonctionnalités.
Il sera obligatoire de tester les codes produits à chaque étape à l'intérieur d'une classe `Test`.

Ce TP est un **travail individuel** à réaliser. Il se pourrait qu'il soit évalué à la fin de la dernière séance, sur la base du travail réalisé tout au long des séances.

[A LIRE] Préliminaire : Consignes

Contenu du sujet

Vous allez concevoir un jeu de bataille navale, en allant de la modélisation du jeu en diagramme de classe jusqu'à l'implantation concrète du code.

Le sujet vous guide à travers différentes étapes à franchir. Celles-ci sont à faire dans l'ordre énoncé par le sujet par soucis de simplicité.

Lisez attentivement le sujet une fois en entier avant de vous lancer dans la programmation.

Dépot Git

Pour que l'on puisse suivre votre progression, vous allez chacun travailler dans un dépot Git. Vous initierez un dépôt Git local dans le répertoire de votre TP (cf Etape 0).
Vous veillerez à commiter résultats régulièrement. Vos messages de *commit* devront être explicites et commencer par le nom de l'étape (cf sujet ci-dessous) concernée par le commit.
Par exemple, pour l'étape 1 : `git commit -m "Etape 1 : MESSAGE"` où MESSAGE est une description du travail poussé dans le dépôt.

Le dépôt Git a pour but de permet de facilement suivre votre progression, et de voir votre régularité tout au long des séances à l'aide de la commande `git log`. Autrement dit, on ne s'attend pas à voir un *commit* qui ajoute 99% du code à 10 minutes de la fin de la dernière séance.

Test unitaires et documentation

Toute classe et ses méthodes devront être testées dans une classe `Test` afin de valider chacune des fonctionnalités.

Vous veillerez à commenter votre code de manière concise mais intelligible.

Etape 1 : Modélisation du programme

Dans cette première étape, on vous propose les grandes lignes d'une implantation du jeu. Les principaux attributs et méthodes associés au comportement du programme sont mentionnés, mais pas forcément de manière exhaustive. Vous devrez aussi réfléchir au(x) constructeur(s) de chaque classe, quels seront leurs paramètres etc.

Règles du jeu

Le but est de réaliser un jeu de Bataille Navale à deux joueurs, en ligne de commande.

Dans ce jeu, chaque joueur possède 5 navires de taille différente (2 à 5 emplacements intacts), qu'il va placer sur son plateau. Chaque joueur ne connait que les emplacements de ses propres bateaux au début du jeu. Chaque joueur possède donc en réalité deux plateaux : le sien récapitulant les emplacements et état de ses bateaux, ainsi qu'un plateau vierge sur lequel il va deviner le placement de la flotte adverse. Nous utiliserons le fait que notre programme a une vue sur les plateaux des deux joueurs (comme une sorte de Maître du Jeu) pour déduire de la grille d'exploration d'un joueur à partir de celle de l'autre. Ainsi, un joueur n'a besoin que de son propre plateau. C'est le jeu qui lui affichera ses frappes passées en les déduisant du plateau adverse.

Une fois que chaque joueur a placé ses bateaux sur sa grille (grille de même taille entre les deux joueurs), les deux joueurs annoncent à tour de rôle à l'autre joueur une case de la grille sur laquelle il tire. Si un bateau se trouve sur la case proposée, le jeu annonce alors "Touché !", mettant à jour le bateau ainsi touché du joueur impacté. Si le bateau n'a plus d'emplacement intact, le jeu annonce en plus "Bravo, vous avez coulé un `X` adverse !", où `X` est le nom du bateau coulé. Si le tir ne touche aucun bateau, le jeu annonce alors "Râté !". Le jeu s'arrête lorsque l'un des deux joueurs n'a plus de bateau hors d'eau.

Il y aura un joueur humain et un joueur ordinateur IA (avec possiblement différentes difficultés dans des versions ultérieures du jeu). Le jeu se déroule donc en 2 phases :

- la phase de placement : L'utilisateur doit entrer les positions de ses 5 navires. Les navires de l'ordinateur sont également placés aléatoirement pour le moment. Il ne peut y avoir deux bateaux qui se touchent.
- la phase de jeu : Le joueur et l'ordinateur alternent les frappes jusqu'à la fin du jeu.

Le jeu implantera les types de navires suivants :

- 1x Torpilleur (To) - taille 2
- 2x Croiseurs (Cr) - taille 3
- 1x Cuirassé (Cu) - taille 4
- 1x Porte-Avion(PA) - taille 5

Spécifications du programme

Le programme devra répondre aux spécifications suivantes.

Configuration La **configuration** du jeu sera déléguée **à une classe abstraite**, c'est à dire non instantiable, possédant deux attributs.
Le premier rassemblant toutes les spécifications des bateaux : identifiant unique, nom et taille. Cet attribut sera statique et non redéfinissable dans une classe fille. Cela permettra à la classe concernée d'être utilisée comme point d'accès aux données concernant les bateaux.

```
private static final String[][] bateaux =
{
    {"1", "Porte-avions", "5"},
    {"2", "Cuirasse", "4",},
    {"3", "Croiseur", "3"},
    {"4", "Croiseur", "3"},
    {"5", "Torpilleur", "2"}
};
```

Sur le même principe, le deuxième attribut sera la taille de la grille. Pour l'instant, nous fixerons la taille à 10 (indices allant donc de A à J pour les colonnes et de 0 à 9 pour les lignes).
Il faudra ajouter à cette classe des méthodes (**statiques** elles aussi) pour accéder au tableau de bateaux, à la description d'un bateau en particulier, connaître le nombre de bateaux et la taille de la grille.

Plateau et cases Un **plateau** est composé de **cases**.

Une **case** possède un numéro associé de colonne et de ligne (donc entre 0 et `Configuration.taille`-1), l'identifiant du bateau éventuellement dessus (0 si pas de bateau), ainsi qu'un attribut booléen pour savoir si la case a été touché par l'adversaire ou non. Les méthodes associées seront de base des *getters* des attributs.

Un **plateau** est ainsi un tableau à deux dimensions de cases, associé à un tableau de bateaux. Les colonnes sont libellées par des lettres majuscules, et les lignes par des numéros commençant par 0.

On doit pouvoir afficher le plateau du joueur (c'est à dire ses bateaux sur sa grille et leur état), mais aussi où en sont les deux joueurs de la découverte du plateau de leur adversaire (c'est à dire les frappes réalisées avec succès et sans succès). Pour cela, nous aurons besoin de deux méthodes d'affichage : (1) une qui utilisera `this` pour afficher le plateau du joueur appelant, (2) une autre méthode qui va prendre deux `Joueurs` en paramètre : un joueur "courant" et son adversaire. Cette méthode affichera successivement le plateau de frappes des deux joueurs à partir du plateau de l'autre (par le propriétés des cases du plateau adverse)
D'autres méthodes telles que l'ajout d'un bateau à un plateau, vérifier si un tableau de `Cases` d'un bateau ne dépasse pas le plateau, vérifier si un tableau de cases ne touche pas un autre bateau seront nécessaires à l'initialisation du plateau.

Vous devrez sûrement ajouter d'autres méthodes à votre code pour le rendre plus modulaire lors de la programmation du jeu.

Bateaux Un bateau possède un identifiant unique, un nom ainsi qu'un tableau de cases sur lequel il se trouve. Pour les méthodes, nous aurons besoin de savoir si le bateau est coulé ou non, l'accès à une des cases du tableau soit par la position dans le tableau, soit et par ses coordonnées x et y, une méthode qui retourne la taille du bateau, ainsi que des *getters* sur les attributs.

Joueurs Les fonctionnalités communes associées aux joueurs (qu'il soit humain ou ordinateur) seront regroupées dans une classe abstraite. Un joueur a un nom, un plateau ainsi qu'un compteur de frappes totales, frappes réussies, et nombre de bateaux adverses coulés. On stocke également la case de la dernière frappe lancée par ce joueur, permettant un affichage spécifique pour permettre une continuité de stratégie. Pour permettre la saisie des coordonnées de frappe ou du nom, un attribut de type `BufferedReader` sera également encapsulé [LIEN.

Trois méthodes abstraites sont déléguées aux classes filles spécialisées : placer les bateaux, tirer sur un joueur adverse et initialiser le nom du joueur.

D'autres méthodes sont directement implantées comme les *getters* des différents attributs, la mise à jour de la dernière case frappée, ainsi que l'incrémentation des compteurs. Vous coderez également une fonction qui récapitule les statistiques du joueur : frappes totales, frappes réussies etc

Joueur humain C'est une classe fille de `Joueur`, qui implante les 3 méthodes abstraites ci-dessus. La procédure de placement des bateaux sera réalisée en saisie clavier par le joueur. Il faudra alors vérifier que les placements proposés par le joueur sont corrects (toutes les cases des bateaux sur la grille, pas deux bateaux qui se touchent). La procédure de frappe sera également gérée au clavier. On fera en sorte de redemander une saisie en cas de coordonnée incorrecte, ou d'une case où une frappe a déjà été réalisée.

Joueur ordinateur Même principe que pour le joueur humain, sauf que le placement des bateaux est réalisé aléatoirement, ainsi que la décision des coordonnées de la frappe. Plus tard, nous pourrons implanter un algorithme de jeu plus intelligent.

Menu Le lancement du jeu est géré par un **menu**, qui devra laisser au joueur le choix entre jouer, afficher les règles du jeu ou quitter le programme. La gestion des saisies clavier du joueur pourront être faite à partir de la classe `BufferedReader` du package `java.io.reader` [LIEN].

Partie Une **partie** se compose de deux joueurs. On utilisera un entier pour savoir quel joueur sera le prochain à jouer. Le vainqueur d'une partie est donné sous la forme d'une chaîne de caractère affichant le nom du joueur vainqueur. Les méthodes attendues dans cette classe sont : initialiser une partie (placement des bateaux par les deux joueurs), jouer une partie (chaque joueur joue à tour de rôle), ainsi que vérifier après chaque tir si la partie est terminée ou non et réaliser l'affichage si c'est le cas. Le tirage au sort de quel joueur joue en premier est aussi réalisé à l'aide d'une fonctionnalité de cette classe.

Programme principal Le programme principal est composé d'un menu, qui soit affiche les règles, soit lance une partie, soit quitte le programme.

Classes supports Certaines classes utilitaires vous sont fournies pour vous aider à manipuler la gestion des coordonnées des cases entre lettres et chiffres :

```
package Support
public class TraitementCoordonnee {

    /**
     * Transforme une coordonnee lettre en nombre
     * Attention ici, on considere les grilles de taille 10 seulement. A
        changer si jamais la taille de la grille devient potentiellement
        plus grande
     */
    public static String CoordonneeLettreversNombre(String coord) {
        return String.valueOf(new String("ABCDEFGHIJ").indexOf(coord));
    }

    /**
     * Transforme une coordonnee nombre dans son pendant en lettre
     * Meme remarque que precedemment
     */
    public static String coordonneeNombreVersLettre(int coord) {
        String[] lettres = { "A", "B", "C", "D", "E", "F", "G", "H", "I",
            "J" };
        return lettres[coord];
    }
```

Ces classes seront regroupées dans un paquet spécifique `Support`. Sur le même principe, vous pourrez créer tout une classe d'aide à l'affichage de votre jeu (couleurs pour menu et plateaux etc). Le jeu étant en console, essayez de générer des affichages simples mais propres.

Etape 0 : Modélisation par diagramme de classe

Avant de se lancer dans l'implantation du jeu, vous allez tout d'abord passer par une phase de modélisation et d'initialisation de votre répertoire Git.

Travail 1 : Diagramme de classe

Avant de vous lancer, entraînez vous à écrire le diagramme de classe associé aux spécifications ci-dessus. Mettez bien les visiblités (*protected*, *public*, *private*) aux attributs et méthodes.

Ce diagramme vous servira à la fois de point de départ pour l'implantation, mais aussi de vue globale sur le projet pour vous guider tout au long des séances. Vous pouvez dessiner ce diagramme en ligne (par exemple sur draw.io), ou plus simplement et plus rapidement sur papier.

Vous **veillerez à le maintenir à jour** au fur et à mesure des progrès de votre implantation du jeu.

Travail 2 : Mise en place du projet et du dépôt Git

Créer dans votre répertoire personnel un dossier `BatailleNavale`. C'est dans ce répertoire que vous allez travailler.

A partir de ce dossier, initialiser dedans un dépôt Git en exécutant la commande :

```
$: git init
```

Créez un fichier `README.md` détaillant comment lancer votre programme (il est également fortement recommandé de fournir un `Makefile`) et quels sont les progrès de votre programme en terme d'étapes. Faites un premier commit de ce fichier :

```
$: git add README.md
$: git commit -m 'Commit initial du README'
```

Créez un dossier `src/` dans `BatailleNavale`, qui servira à stocker vos sources. Créez une classe `src/Test.java` qui servira à tester **TOUTES** les fonctionnalités de chaque classe.

Vous êtes maintenant prêt à commencer l'implantation du jeu. Nous allons faire cela étape par étape. Pour rappel, vous devez **tester toutes les fonctionnalités de vos classes** dans la classe `Test`.

Etape 1 : Configuration et Menu

Configuration Commencez par écrire la classe `Configuration.java` en suivant les spécifications énoncées précédemment.

Quelques consignes supplémentaires :

— Pour la méthode qui retourne la configuration d'un bateau, levez une exception si l'indice passé en paramètre est `OutOfBound`

— Vous devriez avoir 4 méthodes et deux attributs

Ajoutez des tests dans votre classe `Test.java` pour valider votre implantation.

Menu On s'intéresse maintenant à la classe `Menu`. Implanter les trois méthodes demandées, ainsi que la méthode permettant la saisie des options par le joueur. La gestion des saisie peut être faite avec :

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.regex.Pattern;

private BufferedReader in = new BufferedReader(new
    InputStreamReader(System.in));
String input = "";
do {
       System.out.print("Saisissez votre choix : [1, 2 ou 3] ");
       try {
                  input = in.readLine();
             } catch (java.io.IOException e) {
                  System.out.println("Une erreur est survenue : " + e);
             }
         } while (!Pattern.matches("[123]", input));
```

La classe `java.util.regex.Pattern` est très utile pour vérifier que les saisies respectent une expression régulière simple. Ce sera souvent le cas pour nous (saisie des coordonnées, des choix de menus etc)

Vous pouvez, pour rendre vos affichages propres, nettoyer la console avec

```
 try {
             new ProcessBuilder("bash", "-c",
                "clear").inheritIO().start().waitFor();
       } catch (final Exception e){
             System.out.println("Erreur : " + e);
         }
    }
```

Après avoir testé ces fonctionnalités dans votre classe de test, vous pouvez créer votre classe `src/BatailleNavale.java` qui met en oeuvre le menu (pour le moment sans lancer de partie évidemment). Vous n'aurez ensuite plus qu'à la compléter au fur et à mesure avec vos fonctionnalités. Faites un affichage simple mais néanmoins lisible.

N'oubliez pas de faire un (ou plusieurs) *commit* de vos changements dans votre répertoire Git.

Etape 2 : Cases

Implantez la classe `Case` qui sera une des composantes du plateau.

Vous devriez avoir 4 attributs ainsi qu'un constructeur avec deux entiers représentant les coordonnées x et y de la case. Pour rappel, par défaut il n'y a pas de bateau sur la case (initialisation de l'attribut associé à la valeur 0).
Codez les fonctions associées à ces attributs qui vous semblent pertinentes.

Avant de passer à l'étape suivante, faites des tests et valider vos changements avec un *commit*.

Etape 3 : Bateaux

Nous allons désormais implanter la classe `Bateau`, nécessaire pour le plateau.

En suivant les spécifications du programme, vous devriez avoir 3 attributs. A priori, vous aurez besoin d'un constructeur paramétré par ces trois attributs.

Codez ensuite les méthodes d'accès aux attributs, puis ajouter la méthode déterminant si un bateau est coulé ou non. Pour cela, utilisez les propriétés des cases sur lesquelles se trouve le bateau concerné.

Au total, vous devriez avoir 7 méthodes dont 3 accesseurs.

Avant de passer à l'étape suivante, écrivez les tests associés à la classe, et commitez vos changements.

Etape 4 : Plateau

Voici la première classe conséquente que vous allez devoir implanter.

Fonctionnalités basiques Commencez par ajouter les deux attributs de la classe. Codez une fonction qui initialise le tableau de case à l'aide d'une méthode. Vous appelerez le constructeur paramètré de `Case` avec les coordonnées x et y. Appelez ensuite cette méthode dans le constructeur de la classe, et profitez en pour intialiser le tableau de bateaux.

Codez la méthode qui permet d'accéder à une case du plateau, puis l'ajout d'une bateau au tableau de bateaux. Attention, en plus de l'ajouter au tableau, il faudra mettre à jour l'id des cases concernées par ce bateau dans le tableau de cases avec l'id du bateau ainsi ajouté.

Codez la méthode qui vérifie que pour un tableau de cases donné (correspondant à un bateau), ce tableau aurait ou non des cases voisines dans le plateau. Testez bien cette méthode dans votre classe `Test` car elle est critique pour l'initilisation du jeu.

Codez une méthode qui vérifie que pour un tableau de cases donné (correspondant à un bateau), le tableau ne dépasse pas du tableau. Testez bien cette méthode (avec des cas limites) dans votre classe `Test` car elle est critique pour l'initilisation du jeu.

Vous aurez peut-être besoin d'autres méthodes (par exemple pour automatiquement générer un tableau de cases pour un bateau étant donné sa taille, une case départ et une direction donnée). On reviendra là-dessus plus tard.
En attendant, codez les accesseurs aux attributs, ainsi qu'un accesseur à un bateau particulier en fonction de son id.

Affichages des plateaux Nous allons ici nous intéresser aux fonctions d'affichage du plateau de la classe.

Tout d'abord, vous allez coder une méthode qui permet d'afficher le plateau d'un joueur, c'est à dire le contenu du tableau de cases de `this`. Faites un affichage propre, avec les colonnes libélées avec des lettres et les lignes libellées avec des chiffres. Vous pouvez afficher les cases de bateaux coulées avec un code couleur ou graphique spécifique. Testez bien votre méthode dans votre classe `Test`.

Maintenant, nous allons initialiser une méthode qui affiche les plateaux de tirs des deux joueurs. Cette méthode prendra en paramètre un `Joueur joueur`, et un `Joueur adversaire`. Comme nous avons besoin d'implanter la classe `Joueur` pour coder cette méthode, nous reviendrons dessus plus tard.

Validez vos changements avec un *commit* avant de passer à la suite.

Etape 5 : Joueur

Codez la classe abstraite `Joueur` telle que décrite dans les spécifications. Vous initialiserez par défaut la valeur de la case du dernier tir à des coordonnées de valeur $(-1, -1)$. Implantez les méthodes non abstraites (les *getters*, le `setter` de la case du dernier tir, les incrémentations des attributs statistiques etc).

Validez vos changements avec un *commit* avant de passer à la suite.

Etape 6 : Joueur humain

Codez la classe représentant le joueur humain. Son constructeur devra faire appel à la méthode d'initialisation de son nom (déclarée abstraite dans la classe mère). Vous pouvez ajouter à la classe mère un attribut de type `BufferedReader` pour les saisies clavier si ce n'est pas déjà fait.

Les méthodes de placement et de tir devront demander des saisies à l'utilisateur, et vérifier la corrections des actions réalisées.

Ce n'est pas trivial, réfléchissez bien pour ça correctement. Testez votre code dans la classe `Test`.

Validez vos changements avec un *commit* avant de passer à la suite.

Etape 7 : Joueur Bot basique

Codez la classe représentant le joueur ordinateur. Son constructeur devra faire appel à la méthode d'initialisation du nom de l'ordinateur qui retournera un nom dont vous définierez une valeur.
Codez ensuite la méthode de placement des bateaux. Utilisez des méthodes déjà implantées dans `Plateau`. Vous pouvez générer des nombres aléatoires avec `Math.random()` en important `import java.util.Random;`.
Ce n'est pas trivial, réfléchissez bien pour ça correctement. Testez votre code dans la classe `Test`.

Codez enfin la méthode qui détermine une case sur laquelle frapper. Cette case doit être valide et non-déjà la cible d'une frappe de la part de l'ordinateur.
Là encore, faites des tests pour vous assurer de la correction de votre code.

Validez vos changements avec un *commit* avant de passer à la suite.

Etape 8 : Première version du jeu

A cet étape, vous avez toutes les fonctionnalités pour pouvoir réaliser une première version fonctionnelle de votre jeu.

La classe Partie

Commencez par implanter la classe représentant une partie. Une partie se joue entre deux joueurs. Il faut d'abord définir un type de joueur pour chaque joueur (on pourra laisser l'utilisateur décider, pour par exemple faire ordinateur vs joueur, ordinateur vs ordinateur ou joueur vs joueur), puis initialiser leur plateur. Le jeu se charge en suite de faire jouer les joueurs à tour de rôle jusqu'à ce que l'un deux ait gagné. Un affichage des résultats et statistiques de la partie est réalisé à la fin de la partie.

En vous référant aux spécifications, vous devriez avoir 5 attributs (les joueurs, l'entier pour savoir à quel joueur est le prochain tour, une chaîne représentant le joureur gagnant ainsi qu'un buffer pour les saisies clavier).

Vous aurez besoin de différentes méthodes : (1) initialisation pour créer les joueurs et placer leurs bateaux, (2) de jeu qui boucle entre les deux joueurs jusqu'à ce qu'il y ait un vainqueur, (3) fin de partie pour afficher les résultats et statistiques des joueurs (fonctionnalités de la classe `Joueur`), (4) vérifier si un des deux joueurs a gagné, (5) éventuellement tirer au sort quel joueur commence à joueur.

Vous pourrez ajouter d'autres méthodes que vous jugez nécessaire, avec l'objectif de rendre votre code le plus modulaire possible (séparer des fonctionnalités quand elles sont indépendantes).

N'oubliez pas de tester votre code avant de passer à la classe suivante, ainsi de que de faire un *commit*.

La classe Menu

Implantez la case `Menu` permettant d'afficher un menu de jeu, les règles du jeu, et de demander au joueur ce qu'il souhaite faire.

Programme principal

Le programme principal est composé d'un menu, qui permet de réaliser ses affichages et de récupérer les choix des joueurs dans une boucle infini. Quand une partie est lancée, ce programme se charge de la créer, l'initialiser et la lancer. Quand la partie est terminée, on termine la partie en réalisant les affichages et en quittant la boucle.

Tester le jeu en lancant deux joueurs ordinateur, et vérifiez que tout se déroule bien.

N'oubliez pas de tester tous les casd de figure, et de commiter votre code.

[BONUS] Ajout des radars

Une variante du jeu de bataille navale consiste à implémenter les radars. Au début du jeu, chaque joueur possède deux radars. Avant un de ses tours, un joueur peut décider d'utiliser un de ses radars s'il lui en reste. Il choisit alors une case sur laquelle tirer le radar. Le radar permet de savoir si deux cases aux alentour du tir en haut, en bas, à gauche ou à droite, il y a au moins une case intacte d'un bateau adverse qui s'y trouve. Si oui, un message de succès informe le joueur que le radar a détecté quelque chose. Dans le cas contraire, le joueur est informé que rien n'a été trouvé.

Implantez la fonctionnalité de radar dans le jeu pour tous les joueurs. Vous déciderez pour chaque version de vos joueurs ordinateur (cf ci-dessous), à quel moment il peut utiliser le radar. Cela peut-être de manière aléatoire ou alors concerté dans la stratégie d'exploration.

Testez bien la correction de votre implantation lançant des radars sur une grille de test.

[BONUS] Joueur bot amélioré

On peut imaginer de nombreuses améliorations du programme concernant la manière de jouer de l'ordinateur. Vous pouvez regarder ici pour différentes approches plus ou moins sophistiquées.

— on pourrait décider que l'ordinateur ne tire plus au hasard quand il a touché un bateau pour la première fois. Dans ce cas, il quadrillerait les cases aux alentours de la première touche pour débusquer le bateau adverse. On pourrait imaginer rajouter un tableau de cases pontentielles en fonction de cette frappe réussie, et ensuite l'affiner en explorant les possibles localisations des autres emplacement du bateau chassé en fonction des cases déjà touchées

— Plutôt que de tirer au hasard pour trouver un premier impact sur un bateau, on pourrait quadriller efficacement la grille (par exemple de 5 en 5 tant qu'il reste le porte-avion puis descendre) pour chasser les bateaux en utilisant la connaissance de la taille de ceux-ci. Un quadrillage est le tir en damier (sur les diagonales, écartées de deux colonnes), qui permet de couvrir efficacement les bateaux étant donné qu'ils ne peuvent se toucher. Néanmoins, cette stratégie peut ne pas trouver le bateau de taille 2. Faire une exploration en damier une case sur deux corrige ce problème mais se révèle plus lente pour trouver les autres.

Le choix de la difficulté de l'adversaire ordinateur pourrait être laissée au choix du joueur humain. En fonction de la difficulté choisie, le programme appliquerait la stratégie de jeu adaptée pour l'ordinateur.

Vous ferez en sorte de bien réfléchir aux modifications dans la structure du programme et de les propager dans le diagramme de classe avant de vous lancer dans l'implantation.