DUT Informatique Année Spéciale

2017 – 2018

Bases de la PO

Projet Bataille

Steven DELMARQUE

Louis ERNENWEIN

## Table des matières

[Table des matières 2](#_Toc503398578)

[Introduction 3](#_Toc503398579)

[Diagramme UML 4](#_Toc503398580)

[Tests Unitaires 5](#_Toc503398581)

[Sources 6](#_Toc503398582)

[Bilan 26](#_Toc503398583)

## Introduction

Dans le cadre du cours de BPO de M. POITRENAUD, il nous a été demandé de réaliser le programme suivant. Il s’agit d’une bataille navale non-interactive, affichée en console, au cours de laquelle des bateaux se déplacent et ont divers comportements (soigner, attaquer).

Le projet a été réalisé du 4/12/2017 au 12/01/2018. Nous avons réussi à le faire sans aide extérieure, bien que le code pourrait sans doute être amélioré (polymorphisme, ajout d’interface, organisation).

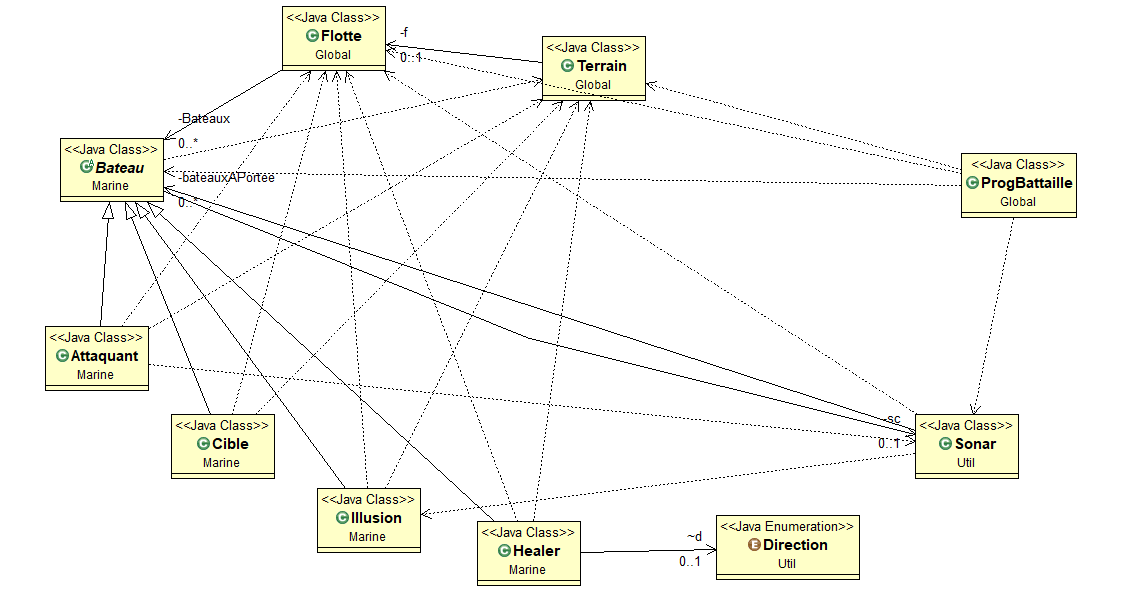
Néanmoins, il fonctionne, et peut, en plus des features demandés dans l’énoncé :

* accueillir un nombre indéfini (mais non infini !) de bateaux
* fonctionner sur n’importe quelle taille de plateau carré
* suivre en direct l’évolution des PV et du sonar des bateaux

Nous vous souhaitons une bonne lecture de ce dossier !

Louis et Steven

## Diagramme UML



## Tests Unitaires

package Test;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import Global.Terrain;

class ProgBattailleTest {

@Test

public void testCreation() { // Test de la création des bateau

int nbBateau = 10;

Terrain t = new Terrain (nbBateau);

assertTrue(t.getFlotte() != null);

assertTrue(t.getFlotte().getBateaux() != null);

assertTrue(t.getFlotte().getBateaux()[0].getNomB() != '\u0000');

assertTrue(t.getFlotte().getBateaux()[0].getSc() != null);

}

@Test

public void testTypeBateau() { // Test de la bonne attribution des types de bateau à l'initialisation

int nbBateau = 10;

Terrain t = new Terrain (nbBateau);

int i;

for (i = 0; i < 3 ; ++i)

assertTrue(t.getFlotte().getBateaux()[i].getNomB() == 'H');

for(i = 3; i < 6 ; ++i)

assertTrue(t.getFlotte().getBateaux()[i].getNomB() == 'A');

for(i = 6; i < t.getFlotte().getNbBateaux() ; ++i)

assertTrue(t.getFlotte().getBateaux()[i].getNomB() == 'C');

}

}

## Sources

ProgBataille.java

**package** Global;

//import Exception.TerrainInitNullException;

/\*\*

\* Programme permettant de simuler une bataille navale, affichée en console.

\* Le programme se déroule sans intervention extérieure

\* Des bateaux parcourent le plateau de jeu à chaque tour.

\* Les Attaquants pourchassent et tirent sur les autres bateaux

\* Les Healer soignent les autres bateaux

\* Les Cibles sont... des cibles !

\* **@author** Al

\*

\*/

**public** **class** ProgBattaille {

**public** **static** **void** main(String[] args) {

**int** nbBateau = 15; // réglable si on veut plus de bateau !

Terrain t = **null**;

t = **new** Terrain(nbBateau);

**int** i = 0;

**while** (i < 300) { // nb de tours de jeu !

**for** (**int** j = 0; j < nbBateau; ++j) {

**if** (t.getFlotte().getBateaux()[j].getNomB() == 'A') { // si bateau attaquant, lui faire détecter le

// bateau le plus proche

t.getFlotte().getBateaux()[j].plusProche(t.getFlotte());

}

**if** (t.getFlotte().getBateaux()[j].EncoreEnVie()) {

t.getFlotte().getBateaux()[j].deplacer(t); // modifie les coordonnées x y de chaque bateau pour

// qu'ils se déplacent d'une case

}

}

t.refresh(); // réinitialise les cases de terrain avec des ~

**for** (**int** j = 0; j < nbBateau; ++j) {

//if (t.getFlotte().getBateaux()[j].EncoreEnVie())

{

t.inclureBateau(t.getFlotte().getBateaux()[j]); // met les nouvelles coordonnées du bateau dans le

// tableau Terrain

}

}

System.***out***.print(t.toString()); // affiche le plateau de jeu

System.***out***.println(*toString*(nbBateau, t)); // affiche le statut des bateaux

*pause*();

++i;

}

}

/\*\*

\* Permet de ralentir l'exécution du programme pour la rendre compréhensible

\*/

**private** **static** **void** pause() { // ralentit l'exécution du programme à chaque tour

**try** {

Thread.*sleep*(300);

} **catch** (Exception e) {

}

}

/\*\*

\* Décrit l'état des bateaux (PV, Bateaux contenus dans le Sonar)

\*

\* **@param** nbBateau

\* le chiffre limitant la boucle d'écriture

\* **@param** t

\* le Terrain contenant les bateaux

\* **@return** le String qui contient la description

\*/

**public** **static** String toString(**int** nbBateau, Terrain t) {

String s = **new** String();

s = "";

s += ("Operation Status");

s += System.*getProperty*("line.separator");

**for** (**int** k = 0; k < nbBateau; ++k) {

**if** (t.getFlotte().getBateaux()[k].EncoreEnVie()) {

s += t.getFlotte().getBateaux()[k].getNomB();

s += t.getFlotte().getBateaux()[k].getNum();

s += " PV = " + (t.getFlotte().getBateaux()[k].getPv());

**for** (**int** l = 0; l < nbBateau; l++)

{

**if** (t.getFlotte().getBateaux()[k].getSc().getBateauxAPortee()[l] != **null**) {

s += " Sonar " + l + " : ";

s += t.getFlotte().getBateaux()[k].getSc().getBateauxAPortee()[l].getNomB();

s += t.getFlotte().getBateaux()[k].getSc().getBateauxAPortee()[l].getNum();

s += " Coord : ";

s += t.getFlotte().getBateaux()[k].getSc().getBateauxAPortee()[l].getX();

s += "-";

s += t.getFlotte().getBateaux()[k].getSc().getBateauxAPortee()[l].getY();

s += " ";

}

}

s += System.*getProperty*("line.separator");

}

}

**return** s;

}

}

Terrain.java

**package** Global;

//import Exception.TerrainInitNullException;

**import** Marine.Bateau;

/\*\*

\* Plateau de jeu représentant l'océan où évoluent les bateaux

\* **@author** Al

\* terrain et terrain 2 contiennent l'ensemble des paires de caractères représentant les cases du tableau.

\* ces paires évoluent en fonction de ce qui les occupent (case vide ou bateau)

\*/

**public** **class** Terrain {

**private** **final** **static** **int** ***TAILLE\_OCEAN*** = 20;

**private** **char**[][] terrain;

**private** **char**[][] terrain2;

**private** Flotte f;

/\*\*

\* Constructeur permettant d'initialiser la Flotte qui crééra les Bateaux

\* **@param** nbBateau contient le nombre de bateaux à initialiser

\*/

**public** Terrain(**int** nbBateau) {

**this**(***TAILLE\_OCEAN***, ***TAILLE\_OCEAN***);

f=**new** Flotte(nbBateau);

}

/\*\*

\* Constructeur du Terrain initial

\* **@param** tLigne contient le nombre de lignes

\* **@param** tColonne contient le nombre de colonnes

\*/

**public** Terrain(**int** tLigne, **int** tColonne) {

terrain = **new** **char**[tLigne][tColonne];

terrain2 = **new** **char**[tLigne][tColonne];

**for** (**int** i = 0; i < tLigne; ++i) {

**for** (**int** j = 0; j < tColonne; ++j) {

**this**.terrain[i][j] = '~';

**this**.terrain2[i][j] = '~';

}

}

}

/\*\*

\* Permet l'affichage du plateau à chaque tour de jeu

\*/

**public** String toString() {

String s = "";

**for** (**int** i = 0; i < ***TAILLE\_OCEAN***; ++i) {

**for** (**int** j = 0; j < ***TAILLE\_OCEAN***; ++j) {

s += " " + **this**.terrain[j][i] + **this**.terrain2[j][i];

}

s += " ";

s += System.*getProperty*("line.separator");

}

**for** (**int** i = 0; i < ***TAILLE\_OCEAN*** + 1 ; ++i)

{

s += " ";

}

s += System.*getProperty*("line.separator");

**return** s;

}

**public** **void** setChar(**int** x,**int** y, **char** newChar) { // setter du tableau terrain

**this**.terrain[x][y]=newChar;

}

**public** **static** **int** getTaille(){ // getter de la taille du terrain

**return** ***TAILLE\_OCEAN***;

}

**public** **char**[][] getTerrain(){

**return** terrain;

}

**public** **char**[][] getTerrain2(){

**return** terrain2;

}

**public** Flotte getFlotte() {

**return** **this**.f;

}

**public** **char** getChar(**int** x, **int** y){ // getter du tableau terrain

**return** **this**.terrain[x][y];

}

/\*\*

\* remplace les ~ des cases correspondants aux bateaux par leur nom et numéro

\* **@param** b le bateau dont les coordonnées doivent être enregistrées

\*/

**public** **void** inclureBateau (Bateau b) {

**this**.getTerrain()[b.getX()][b.getY()]=b.getNomB();

**this**.getTerrain2()[b.getX()][b.getY()]=b.getNum();

}

/\*\*

\* met des ~ sur toutes les cases du tableau

\*/

**public** **void** refresh() { //

**for** (**int** i = 0; i < ***TAILLE\_OCEAN***; ++i) {

**for** (**int** j = 0; j < ***TAILLE\_OCEAN***; ++j) {

**this**.terrain[i][j] = '~';

**this**.terrain2[i][j] = '~';

}

}

}

}

Flotte.java

**package** Global;

**import** Marine.Attaquant;

**import** Marine.Bateau;

**import** Marine.Cible;

**import** Marine.Healer;

**import** Exception.TerrainInitNullException;

/\*\*

\* Classe permettant de gérer l'ensemble des bateaux

\* **@author** Al

\* Bateau[] contient l'ensemble des bateaux créés

\* NbBateau permet de retrouver le nombre de bateaux créés

\*/

**public** **class** Flotte {

**private** Bateau[] Bateaux; // tableau qui contient l'ensemble des bateaux créés

**private** **static** **int** *nbBateaux*;

/\*\*

\* Constructeur permettant de créer nbBateau Bateau dans le tableau de bateau Bateaux

\* Le 1er 1/3 des bateaux créés seront des Healer

\* Le 2e 1/3 des bateaux créés seront des Attaquant

\* Le reste des bateaux créés seront des Cible

\* **@param** nbBateau indique le nombre de bateau à créer

\*/

**public** Flotte(**int** nbBateau) // crée une flotte composé à un tiers d'hopitaux, d'un tiers d'attaquant et de 4/10 de cible

//throws TerrainInitNullException

{

**this**.*nbBateaux*=nbBateau;

Bateaux= **new** Bateau[nbBateau];

**int** i;

**for**( i=0;i<(nbBateau/3);++i) {

Bateaux[i]=**new** Healer(nbBateau);

}

**for**(i=(nbBateau/3);i<(2\*(nbBateau/3));++i) {

Bateaux[i]=**new** Attaquant(nbBateau);

}

**for**(i=(2\*(nbBateau/3));i<nbBateau;++i) {

Bateaux[i]= **new** Cible(nbBateau);

}

/\* for (i = 0 ; i < nbBateau ; ++i)

{

if (Bateaux[i].getSc() == null)

System.out.println("prout");

} \*/

}

**public** Bateau[] getBateaux() {

**return** Bateaux;

}

**public** **int** getNbBateaux() {

**return** *nbBateaux*;

}

}

Bateau.java

**package** Marine;

**import** java.util.Random;

**import** Global.Flotte;

**import** Global.Terrain;

**import** Util.Sonar;

**import** Exception.TerrainInitNullException;

/\*\*

\* Classe abstraite représentant une entité Bateau

\* PV\_MAX Points de vie initiaux d'un bateau

\* sc Sonar permettant de repérer les bateaux alentour

\* pv Points de vie actuels d'un bateau

\* x Coordonnée horizontale d'un bateau. Commence à 0

\* y Coordonnée verticale d'un bateau. Commence à 0

\* nomB contient le nom du bateau

\* num contient le numéro du bateau (incrémenté par type)

\* **@author** Al

\*

\*/

**public** **abstract** **class** Bateau {

**private** **final** **static** **int** ***PV\_MAX*** = 100; // résistance du bateau

**private** Sonar sc; // permet de détecter les bateaux alentour

**private** **int** pv; // stocke les PV actuels

**private** **int** x; // coord horizontale

**private** **int** y; // coord verticale

**private** **char** nomB;

**private** **char** num;

/\*\*

\* Constructeur par défaut, utilisé seulement pour les Bateaux Illusion

\* Donne au Bateau ses PV max

\* Donne une position aléatoire au bateau

\*/

**public** Bateau() // Constructeur donnant une position aléatoire au bateau

{

setPv(getPV\_MAX());

Random rd = **new** Random();

setX(rd.nextInt(Terrain.*getTaille*() - 1));

setY(rd.nextInt(Terrain.*getTaille*() - 1));

}

/\*\*

\* Constructeur utilisé pour les bateaux créés par Flotte. Identique au constructeur par défaut mais initialise le sonar

\* **@param** nbBateau permet de donner une taille au tableau de bateau contenu dans sc

\*/

**public** Bateau (**int** nbBateau) // Constructeur donnant une position aléatoire au bateau

{

setPv(getPV\_MAX());

Random rd = **new** Random();

setX(rd.nextInt(Terrain.*getTaille*() - 1));

setY(rd.nextInt(Terrain.*getTaille*() - 1));

setSc(nbBateau);

}

/\*\*

\* Permet de détecter le bateau le plus proche de celui-ci

\* **@param** f contient les informations sur les autres bateaux

\*/

**public** **abstract** **void** plusProche(Flotte f);

/\*\*

\* Permet de modifier les coordonnées d'un bateau, selon des règles propres à chaque sous-type de bateau

\* **@param** t contient les informations sur le terrain et les autres bateaux

\*/

**public** **abstract** **void** deplacer(Terrain t); // methode abstraite utilisée dans les classes dérivées de Bateau

/\*\*

\* Vérifie que les PV d'un Bateau sont supérieurs à 0

\* **@return** true si les PV sont strictement supérieurs à 0

\*/

**public** **boolean** EncoreEnVie() {

**return** **this**.pv > 0;

}

/\*\*

\* Remplace le nomB et num d'un bateau par 'X'

\*/

**public** **void** isDead() {

setNomB('X');

setNum('X');

}

**public** Sonar getSc() {

**return** sc;

}

**public** **void** setSc(**int** nbBateau) {

**this**.sc = **new** Sonar (nbBateau);

}

**public** **char** getNum() {

**return** **this**.num;

}

**public** **void** setNum(**char** n) {

**this**.num = n;

}

**public** **char** getNomB() {

**return** **this**.nomB;

}

**protected** **void** setNomB(**char** nom) {

**this**.nomB = nom;

}

**public** **int** getX() {

**return** **this**.x;

}

**protected** **void** setX(**int** newX) {

**this**.x = newX;

}

**public** **int** getY() {

**return** **this**.y;

}

**protected** **void** setY(**int** newY) {

**this**.y = newY;

}

/\*\*

\* // Permet de déplacer un bateau sur une autre case

\* **@param** newX nouvelle coordonnée horizontale

\* **@param** newY nouvelle coordonnée verticale

\*/

**public** **void** Bouger(**int** newX, **int** newY) {

**this**.x=newX;

**this**.y=newY;

}

**protected** **void** setPv(**int** i) {

**this**.pv=i;

}

**public** **int** getPv() {

**return** **this**.pv;

}

**public** **int** getPV\_MAX() {

**return** **this**.***PV\_MAX***;

}

}

Attaquant.java

**package** Marine;

**import** java.util.Random;

**import** Global.Flotte;

**import** Global.Terrain;

**import** Util.Sonar;

**import** Exception.TerrainInitNullException;

**public** **class** Attaquant **extends** Bateau {

**private** **final** **static** **int** ***DEGATS*** = 10;

**private** **int** distanceAttaque = 3;

**private** **int** indicePlusProche;// attention a la distance differente de 0

**private** **static** **char** *nAtq* = '1';

/\*\*

\* Constructeur du bateau Attaquant.

\* **@param** nbBateau nombre de bateaux

\*/

**public** Attaquant(**int** nbBateau)

{

**super**(nbBateau);

setNomB('A');

setNum(*nAtq*);

setDistanceAttaque(3);

*nAtq*++;

**int** indicePlusProche = -1;

}

/\*\*

\* Méthode permettant d'obtenir après annalyse du tableau du sonar le bateau le plus proche de celui ci.

\* si deux bateaux sont à la même distance il mémorise le dernier verifié.

\* **@param** f la flotte comprenant la totalité des bateaux.

\*/

@Override

**public** **void** plusProche(Flotte f) {

**this**.getSc().detection(f, **this**);

**double** distance = getSc().getPortee();

**for** (**int** i = 0; i < f.getNbBateaux(); ++i) {

**if** (**this**.getSc().getBateauxAPortee()[i] != **null**) {

**if** ((f.getBateaux()[i].getY() != **this**.getY()

&& f.getBateaux()[i].getX() != **this**.getX())

&& distance >= Math.*sqrt*(Math.*pow*(2, (f.getBateaux()[i].getX() - **this**.getX()))

+ Math.*pow*(2, (f.getBateaux()[i].getY() - **this**.getY())))) {

**this**.indicePlusProche = i;

distance = Math.*sqrt*(Math.*pow*(2, (f.getBateaux()[i].getX() - **this**.getX()))

+ Math.*pow*(2, (f.getBateaux()[i].getY() - **this**.getY())));

}

}

}

}

/\*\*

\*

\* **@return** int distance d'attaque d'un bateau

\*/

**public** **int** getDistanceAttaque() {

**return** distanceAttaque;

}

/\*\*

\* Méthode permettant de changer la distance d'attaque d'un bateau.

\* **@param** a nouvelle distance d'attaque

\*/

**public** **void** setDistanceAttaque(**int** a) {

**this**.distanceAttaque = a;

}

/\*\*

\* Une méthode qui permet de faire des dégats aux bateaux présents sur la

\* case du bateau ciblé.

\*

\* **@param** t

\* prend en paramètre le terrain.

\*/

**public** **void** attaque(Terrain t) {

**for** (**int** i = 0; i < t.getFlotte().getNbBateaux(); ++i)

**if** (distanceAttaque >= (Math.*sqrt*(Math.*pow*(2, (t.getFlotte().getBateaux()[i].getX() - **this**.getX()))

+ Math.*pow*(2, (t.getFlotte().getBateaux()[i].getY()

- **this**.getY()))))&& t.getFlotte().getBateaux()[i]!=**this**) {

**if** (t.getFlotte().getBateaux()[i].getX() == t.getFlotte().getBateaux()[indicePlusProche].getX()

&& t.getFlotte().getBateaux()[i].getY() == t.getFlotte().getBateaux()[indicePlusProche].getY()

&& t.getFlotte().getBateaux()[i].getPv() > 0) {

t.getFlotte().getBateaux()[i].setPv(t.getFlotte().getBateaux()[i].getPv() - ***DEGATS***);

**if** (!t.getFlotte().getBateaux()[i].EncoreEnVie())

t.getFlotte().getBateaux()[i].isDead();

}

}

}

/\*\*

\* Une méthode déplacer qui en fonction de la présence de bateaux autour du

\* bateau attaquant choisi de façon de se déplacer différente.

\*

\* **@param** t

\* prend en paramètre le terrain.

\*/

**public** **void** deplacer(Terrain t) {

Random r = **new** Random();

**int** i;

**if** (indicePlusProche == -1) {// si pas de bateaux Ã  portÃ©e

// ,deplacement random

**do** {

i = r.nextInt(3) - 1;// donne un random de -1 ,0 ou1

} **while** (**this**.getX() + i < 0 || **this**.getX() + i > t.*getTaille*() - 1);

**this**.setX(**this**.getX() + i);

**do** {

i = r.nextInt(3) - 1;// donne un random de -1 ,0 ou1

} **while** (**this**.getY() + i < 0 || **this**.getY() + i > t.*getTaille*() - 1);

**this**.setY(**this**.getY() + i); // est remplacable par deriver

} **else** {

**if** (t.getFlotte().getBateaux()[indicePlusProche].getX() > **this**.getX())

**this**.setX(**this**.getX() + 1);

**else** **if** (t.getFlotte().getBateaux()[indicePlusProche].getX() < **this**.getX())

**this**.setX(**this**.getX() - 1);

**else** **if** (t.getFlotte().getBateaux()[indicePlusProche].getX() == **this**.getX())

;

**if** (t.getFlotte().getBateaux()[indicePlusProche].getY() > **this**.getY())

**this**.setY(**this**.getY() + 1);

**else** **if** (t.getFlotte().getBateaux()[indicePlusProche].getY() < **this**.getY())

**this**.setY(**this**.getY() - 1);

**else** **if** (t.getFlotte().getBateaux()[indicePlusProche].getY() == **this**.getY())

;

**this**.attaque(t);

indicePlusProche = -1;

}

}

}

Healer.java

**package** Marine;

**import** java.util.Random;

**import** Global.Flotte;

**import** Global.Terrain;

**import** Global.ProgBattaille;

**import** Util.Direction;

**import** Util.Sonar;

**import** Exception.TerrainInitNullException;

**public** **class** Healer **extends** Bateau {

**private** **int** heal;

Direction d = Direction.***NORD***;

**private** **static** **char** *nHeal* = '1';

/\*\*

\* Constructeur du bateau Healer.

\*

\* **@param** nbBateau nombre de bateaux

\*/

**public** Healer(**int** nbBateau) {

**super**(nbBateau);

heal = 50;

setNomB('H');

setNum(*nHeal*);

*nHeal*++;

}

/\*\*

\* Ajoute la valeur de heal aux PV de Bateau b

\*

\* **@param** b le bateau soigné

\*/

**public** **void** heal(Bateau b) {

**if** (b.getPv() > 0)

b.setPv(b.getPv() + heal);

**if** (b.getPv() > b.getPV\_MAX())

b.setPv(getPV\_MAX());

}

/\*\*

\* Une méthode déplacer qui donne un déplacement aléatoire au bateau hopital

\*

\* **@param** t prend en paramètre le terrain.

\*/

**public** **void** deplacer(Terrain t) {

**while** (getX() + d.getDx() < 0 || getX() + d.getDx() >= t.*getTaille*() || getY() + d.getDy() >= t.*getTaille*()

|| getY() + d.getDy() < 0) {

d = d.deriver(5);

}

**this**.Bouger(getX() + d.getDx(), getY() + d.getDy());

**for** (**int** i = 0; i < t.getFlotte().getNbBateaux(); i++) {

**if** (**this**.getX() == t.getFlotte().getBateaux()[i].getX()

&& **this**.getY() == t.getFlotte().getBateaux()[i].getY()

&& (**this**.getNomB() != t.getFlotte().getBateaux()[i].getNomB()

&& **this**.getNum() != t.getFlotte().getBateaux()[i].getNum()))

heal(t.getFlotte().getBateaux()[i]);

}

}

@Override

**public** **void** plusProche(Flotte f) {

// **TODO** Auto-generated method stub

}

}

Cible.java

**package** Marine;

**import** java.util.Random;

**import** Global.Flotte;

**import** Global.Terrain;

**import** Exception.TerrainInitNullException;

**public** **class** Cible **extends** Bateau {

**private** **boolean** descend = **true**; // controleur de direction

**private** **boolean** droite = **true**;

**private** **static** **char** *nCible* = '1'; // donne son numéro de cible à chaque

// cible créée

/\*\*

\* Constructeur du bateau Cible

\*

\* **@param** nbBateau nombre de bateaux

\*/

**public** Cible(**int** nbBateau)

{

**super**(nbBateau);

setNomB('C');

setNum(*nCible*);

*nCible*++;

}

/\*\*

\* Une méthode déplacer qui fait faire au bateau le déplacement en

\* boustrophédon.

\*

\* **@param** t prend en paramètre le terrain.

\*/

**public** **void** deplacer(Terrain t) { // déplacement en boustrophédon

**if** (getX() == 0 && getY() == Terrain.*getTaille*() - 1 && descend && !droite) {

droite = **true**;

**this**.Bouger(getX() + 1, getY()); // déplacement vers la droite

} **else** **if** (getX() == Terrain.*getTaille*() - 1 && getY() == 0 && !descend && droite) {

droite = **false**;

**this**.Bouger(getX() - 1, getY());

} **else** **if** (descend) {

**if** (getX() < Terrain.*getTaille*() - 1 && droite) {

**this**.Bouger(getX() + 1, getY());

} **else** **if** (getX() == Terrain.*getTaille*() - 1 && droite && getY() != Terrain.*getTaille*() - 1) {

**this**.Bouger(getX(), getY() + 1);

droite = **false**;

} **else** **if** (getX() > 0 && !droite) {

**this**.Bouger(getX() - 1, getY());

} **else** **if** (getX() == 0 && !droite) {

**this**.Bouger(getX(), getY() + 1);

droite = **true**;

} **else** **if** (getX() == Terrain.*getTaille*() - 1 && getY() == Terrain.*getTaille*() - 1) {

descend = **false**;

droite = **false**;

**this**.Bouger(getX() - 1, getY());

}

} **else** {

**if** (getX() > 0 && !droite) {

**this**.Bouger(getX() - 1, getY());

} **else** **if** (getX() == 0 && !droite && getY() > 0) {

**this**.Bouger(getX(), getY() - 1);

droite = **true**;

} **else** **if** (getX() < Terrain.*getTaille*() - 1 && droite) {

**this**.Bouger(getX() + 1, getY());

} **else** **if** (getX() == Terrain.*getTaille*() - 1 && droite) {

**this**.Bouger(getX(), getY() - 1);

droite = **false**;

} **else** **if** (getX() == 0 && getY() == 0) {

descend = **true**;

droite = **true**;

**this**.Bouger(getX() + 1, getY());

}

}

}

@Override

**public** **void** plusProche(Flotte f) {

// **TODO** Auto-generated method stub

}

}

Illusion.java

**package** Marine;

**import** Exception.TerrainInitNullException;

**import** Global.Flotte;

**import** Global.Terrain;

/\*\*

\* Bateau créé uniquement lors d'une Detection raté. Ce bateau copie les informations du bateau mal détecté

\* **@author** Al

\*/

**public** **class** Illusion **extends** Bateau { // bateau appelé pour permettre au sonar de stocker de fausses coordonnées

/\*\*

\* Constructeur copiant les données du bateau copié

\* **@param** x coordonnée horizontale du bateau copié

\* **@param** y coordonnée verticale du bateau copié

\* **@param** c Nom du bateau copié

\* **@param** num Numéro du bateau copié

\*/

**public** Illusion (**int** x, **int** y, **char** c, **char** num)

{

**super**();

setX(x);

setY(y);

setNomB(c);

setNum(num);

}

@Override

**public** **void** deplacer(Terrain t) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** plusProche(Flotte f) {

// **TODO** Auto-generated method stub

}

}

Sonar.java

**package** Util;

**import** java.util.Random;

**import** Global.Flotte;

**import** Marine.Bateau;

**import** Marine.Illusion;

**public** **class** Sonar { // classe permettant de détecter les bateaux autour d'un

// bateau

**private** **int** precision = 90; // chance que la détection soit juste

**private** **static** **final** **int** ***portee*** = 4; // portee en case du sonar

**private** Bateau[] bateauxAPortee; // contient les bateaux repérés

/\*\*

\* Constructeur du Sonar

\*

\* **@param** nbBateau

\* nombre de bateaux

\*/

**public** Sonar(**int** nbBateau) {

// bateauxAPortee = new Bateau[nbBateau];

setBateauxAPortee(nbBateau);

}

/\*\*

\*

\* **@return** int la portée du Sonar

\*/

**public** **int** getPortee() {

**return** ***portee***;

}

/\*\*

\* Renvoie un tableau de bateaux qui indique quels bateaux sont à portée du

\* Sonar.

\*

\* **@return** un tableau de Bateau

\*/

**public** Bateau[] getBateauxAPortee() {

**return** bateauxAPortee;

}

/\*\*

\* Permet de reinitialiser le tableau de bateaux à portés entre deux utilisations

\* **@param** nbBateau nombre de bateaux

\*/

**public** **void** setBateauxAPortee(**int** nbBateau) {

**this**.bateauxAPortee = **new** Bateau[nbBateau];

}

/\*\*

\* Permet de changer un bateau dans une case du tableau de bateaux à portée.

\* **@param** b un bateau

\* **@param** indiceTab l'indice dans le tableau de bateaux

\*/

**public** **void** setBateauxAPortee\_Case(Bateau b, **int** indiceTab) {

**this**.bateauxAPortee[indiceTab] = b;

}

/\*\*

\* Permet d'analyser le tableaux de bateaux de la flotte pour en obtenir les bateaux les plus proches du bateau.

\* **@param** f la flotte

\* **@param** b un bateau

\*/

**public** **void** detection(Flotte f, Bateau b) {

Random rd = **new** Random();

**int** randPrecision;

**for** (**int** i = 0; i < f.getNbBateaux(); ++i) {

randPrecision = rd.nextInt(100);

**if** (randPrecision < **this**.precision) {

**if** (***portee*** >= Math.*sqrt*(Math.*pow*(2, (f.getBateaux()[i].getX() - b.getX()))

+ Math.*pow*(2, (f.getBateaux()[i].getY() - b.getY())))) {

bateauxAPortee[i] = f.getBateaux()[i];

} **else**

bateauxAPortee[i] = **null**;

}

**else** { // si le radar échoue

// bateauxAPortee[i] = null;

**int** result = rd.nextInt(2); // on regarde quelle erreur donne le

// radar.

//System.out.println("cas de détection rapée");

**switch** (result) { // usage d'un switch permet de plus facilement ajouter des cas supplémentaires !

**case** 0:

failMisplace(f, b, i); // dans ce cas, sonar enregistre un bateau avec de mauvaises coordonnées

//System.out.println("mauvaises coordonnées ");

**break**;

**case** 1: // fail\_Nosee (donc rien à faire)

**break**;

**default**:

**break**;

}

}

}

}

/\*\*

\* simple fonction qui teste si le radar renvoie un résultat vrai ou faux

\* **@param** txDetection

\* **@return** Boolean qui indique si le radar se trompe ou pas

\*/

**public** **boolean** radarFail(**int** txDetection) {

Random rd = **new** Random();

**if** (rd.nextInt(100) > txDetection)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

\* Cas d'échec ou le sonar enregistre les mauvaises coordonnées d'un bateau

\* **@param** f la flotte de bateaux

\* **@param** b un bateau

\* **@param** indice indice du tableau

\*/

**public** **void** failMisplace(Flotte f, Bateau b, **int** indice) {

Random rd = **new** Random();

**int** x, y;

**do** {

x = rd.nextInt(2) - 1; // permet, additionné à - 1, d'avoir un décalage aléatoire de [-1 ; 1]

y = rd.nextInt(2) - 1;

} **while** (x == 0 && y == 0); // évite que l'illusion ait la position du bateau mal scanné

//System.out.print(" " + f.getBateaux()[indice].getNomB() + f.getBateaux()[indice].getNum() + " " + x + "/" + y);

Illusion illu = **new** Illusion(f.getBateaux()[indice].getX() - x, f.getBateaux()[indice].getY() - y,

f.getBateaux()[indice].getNomB(), f.getBateaux()[indice].getNum());

// on crée un bateau de type illusion, qui copie les caractéristiques dubbateau mal détecté, avec un décalage de placement

**this**.getBateauxAPortee()[indice] = illu; // on alloue le bateau aux mauvaises coordonnées à la case indice du tableau du sonar.

}

}

Direction.java

**package** Util;

**import** java.util.Random;

**public** **enum** Direction {

***EST***(1, 0), ***NORD\_EST***(1, 1), ***NORD***(0, 1), ***NORD\_OUEST***(-1, 1), ***OUEST*** (-1, 0), ***SUD\_OUEST***(-1, -1), ***SUD***(0, -1), ***SUD\_EST***(1, -1);

// dï¿½placement relatif de la direction

**private** **final** **int** dx, dy;

**private** Direction(**int** dx, **int** dy) {

**this**.dx = dx; **this**.dy = dy;

}

/\*\* Retourne le déplacement sur l'axe des x correspondant à la

\* direction courante.

\* **@return** le déplacement en x

\*/

**public** **int** getDx() {

**return** dx;

}

/\*\* Retourne le déplacement sur l'axe des y correspondant à la

\* direction courante.

\* **@return** le déplacement en y

\*/

**public** **int** getDy() {

**return** dy;

}

/\*\* Retourne une direction inverse à la direction courante.

\* **@return** la direction opposée

\*/

**public** Direction inverser() {

**return** Direction.*values*()[(**this**.ordinal() + 4) % 8];

}

/\*\* Retourne une direction dérivée par rapport à la

\* direction courante d'un angle alï¿½atoirement choisi et

\* compris entre (-marge \* 45°) et (+marge \* 45°). La dérivation

\* est choisie nécessairement pour être un multiple de 45°.

\* **@param** marge la tolérance

\* **@return** la nouvelle direction

\*/

**public** Direction deriver(**int** marge) {

Random rd = **new** Random();

**int** d = (8 + **this**.ordinal() + (rd.nextInt(marge \* 2 + 1) - marge)) % 8;

**return** Direction.*values*()[d];

}

}

## Bilan