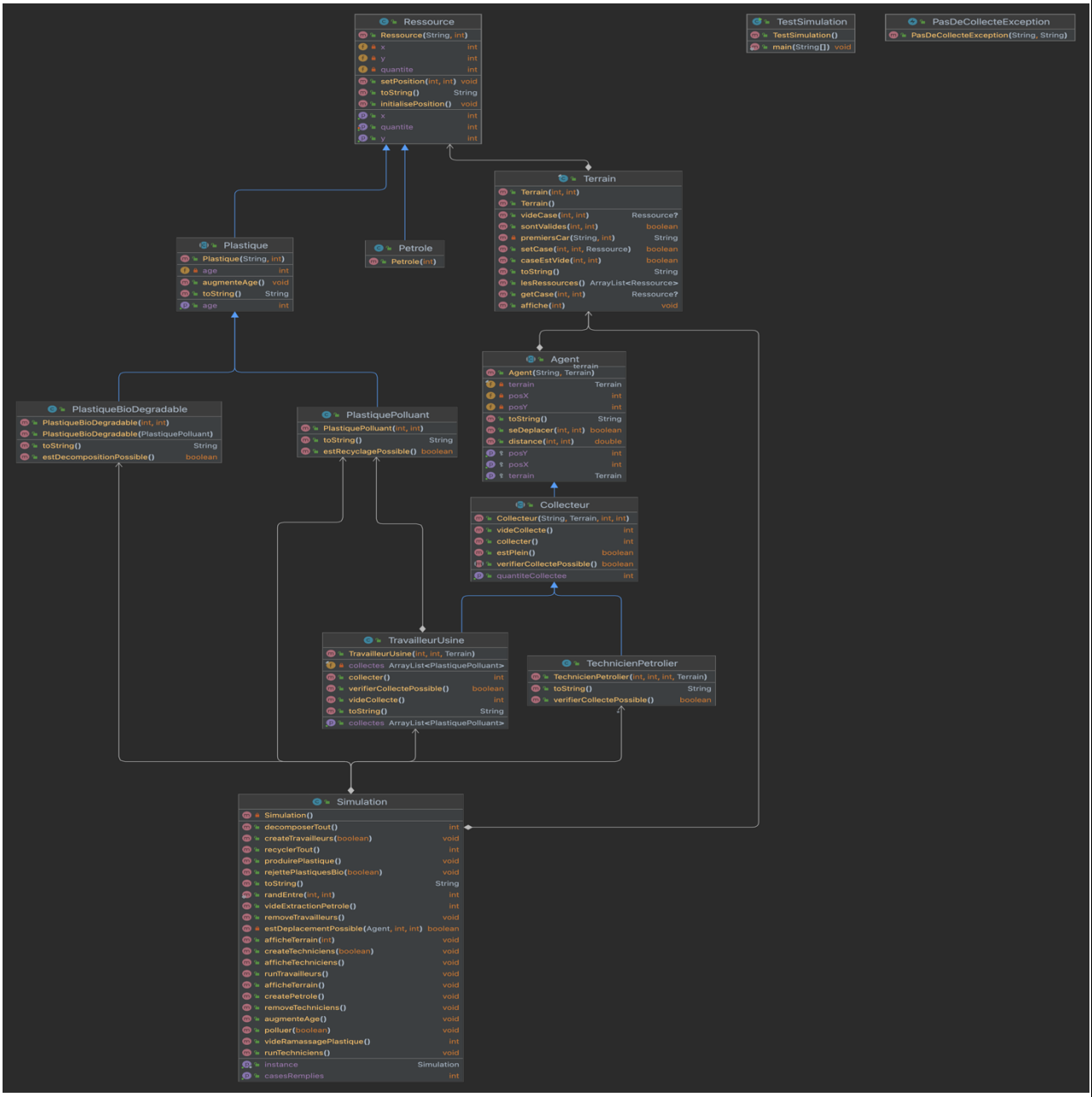
Projet LU2IN002 - 2022-2023

Numéro du groupe de TD/TME : MONO - 4

Nom : BATACHE	*Nom :* MAMLOUK
Prénom : Toufic	*Prénom :* Haya
N° étudiant : 21107810	*N° étudiant :* 21107689

Thème choisi (en 2 lignes max.)
Le terrain s'agit d'une mer dans laquelle on peut trouver des ressources naturelles (pétrole) qu'on extrait, mais aussi c'est un terrain que l'on pollue avec du plastique. Pour cela, le thème de recyclage et de produit biodégradable sont abordés.

Schéma UML des classes vision fournisseur (dessin "à la main" scanné ou photo acceptés)



Checklist des contraintes prises en compte:	*Nom(s) des classe(s) correspondante(s)*
Classe contenant un tableau ou une ArrayList	TrvailleurUsine Simulation
Classe avec membres et méthodes statiques	Agent Simulation
Classe abstraite et méthode abstraite	Collecteur
Interface	Pas d'interface
Classe avec un constructeur par copie ou clone()	Simulation
Définition de classe étendant Exception	PasDeCollecteException
Gestion des exceptions	Collecteur(TechnicienPetrolier, TravailleurUsine) Simulation TestSimulation

Présentation brève de votre projet (max. 10 lignes) : texte libre expliquant en quoi consiste votre projet.

Le projet consiste de répétitions de cycle de simulation.

Au départ, du pétrole est créé et distribué sur le Terrain ainsi que des TechnicienPetrolier et des TravailleurUsine. Ceux-là restent identiques pour tous les cycles.

Avec chaque cycle :
-Les TechnicienPetrolier parcourent le Terrain et extraient du Pétrole pour en produire du PlastiquePolluant.
-Avec cette production, le Terrain se pollue de PlastiquePolluant.
-Pour cela, des TravailleurUsine parcourent le Terrain pour collecter du PlastiquePolluant et le recycler en PlastiqueBioDegradable.
-Le PlastiqueBioDegradable est, à son tour, jeté dans le terrain jusqu'à sa décomposition qui dépend de sa durée de vie.

Les répétitions de cycle s'arrêtent lorsqu'aucune Ressource ne reste présente sur le Terrain

Copier / coller vos classes et interfaces à partir d'ici :

```
/**
 * Représentation des agents qui parcourent le terrain.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
```

```java
 */

public abstract class Agent {
    private static int nbAgentsCrees = 0;
    public final int ident;
    public final String type;
    private final Terrain terrain;
    private int posX;
    private int posY;

    /**
     * Constructeur qui initialise l'agent avec un type et un terrain
     * Affection du numéro ID unique
     *
     * @param type type d'Agent
     * @param t    terrain sur lequel se trouve l'Agent
     */
    public Agent(String type, Terrain t) {
        this.ident = Agent.nbAgentsCrees++;
        this.type = type;
        this.terrain = t;
    }

    /**
     * Permet d'avoir accès au terrain sur lequel se trouve l'Agent
     *
     * @return le terrain sur lequel se trouve l'Agent
     */
    protected Terrain getTerrain() {
        return terrain;
    }

    /**
     * Permet d'avoir accès à la ligne sur laquelle se trouve l'Agent
     *
     * @return la ligne sur laquelle se trouve l'Agent
     */
    protected int getPosX() {
        return posX;
    }

    /**
     * Permet d'avoir accès a la colonne sur laquelle se trouve l'Agent
     *
     * @return la colonne sur laquelle se trouve l'Agent
     */
    protected int getPosY() {
        return posY;
    }

    /**
     * Calcule la distance entre l'Agent et une case donnée
     *
     * @param x abscisse de la case concernée
     * @param y ordonnée de la case concernée
     * @return la distance entre l'Agent et la case concernée
     */
    public double distance(int x, int y) {
        return Math.sqrt(Math.pow(posY - y, 2) + Math.pow(posX - x, 2));
    }

    /**
     * Vérifie si les coordonnées sont présentes sur le terrain et déplace
l'Agent dans la nouvelle case
```

```
     *
     * @param xnew nouvelle abscisse
     * @param ynew nouvelle ordonnée
     * @return si l'Agent a été déplacé dans la nouvelle case
     */
    public boolean seDeplacer(int xnew, int ynew) {
        posX = xnew;
        posY = ynew;
        return true;
    }

    /**
     * Renvoie des informations sur l'Agent
     *
     * @return l'ID et la position de l'Agent
     */
    @Override
    public String toString() {
        return type + "[id:" + ident + "] en position (" + posX + ", " + posY +
").";
    }
}
```

```
/**
 * Généralisation des méthodes communes aux collecteurs.
 * Classe mère des techniciens pétroliers et des travailleurs d'usine.
 * Classes filles sont {@link TechnicienPetrolier} et {@link TravailleurUsine}.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public abstract class Collecteur extends Agent {
    protected final int capaciteDeCollecte;
    protected final int capaciteDeStockage;
    private int qteCollectee;

    /**
     * Constructeur qui initialise l'agent collecteur avec un type et un terrain
     *
     * @param type type d'agent
     * @param t    terrain sur lequel se trouve l'agent
     * @param capaciteDeCollecte capacité de collecte de l'agent
     * @param capaciteDeStockage capacité de stockage de l'agent
     */
    public Collecteur(String type, Terrain t, int capaciteDeCollecte, int
capaciteDeStockage) {
        super(type, t);

        this.capaciteDeCollecte = capaciteDeCollecte;
        this.capaciteDeStockage = capaciteDeStockage;
        this.qteCollectee = 0;
    }

    /**
     * Permet de tester si la collecte est possible ou pas.
     *
     * @return si la collecte est possible
     * @throws Exception si la collecte n'est pas possible pour une raison
fatale
     */
    public abstract boolean verifierCollectePossible() throws Exception;
```

```java
    /**
     * Permet de collecter la ressource qui peut se trouver dans la même case de
l'agent collecteur
     *
     * @return le volume de la collecte
     * @throws Exception si la collecte n'est pas possible pour une raison
fatale
     */
    public int collecter() throws Exception {
        if (!verifierCollectePossible()) {
            return -1;
        }

        Ressource ressource = getTerrain().getCase(getPosX(), getPosY());
        int aCollecter = Math.min(Math.min(ressource.getQuantite(),
capaciteDeCollecte), capaciteDeStockage - getQuantiteCollectee());
        ressource.setQuantite(ressource.getQuantite() - aCollecter);
        if (ressource.getQuantite() == 0) {
            getTerrain().videCase(ressource.getX(), ressource.getY());
        }
        qteCollectee += aCollecter;
        return aCollecter;
    }

    /**
     * Renvoie la quantité collectée par l'agent.
     *
     * @return la quantité collectée
     */
    public int getQuantiteCollectee() {
        return qteCollectee;
    }

    /**
     * Renvoie si le stockage de l'agent est plein ou pas.
     *
     * @return si la quantité collectée est supérieure ou égale à la capacité de
stockage
     */
    public boolean estPlein() {
        return qteCollectee >= capaciteDeStockage;
    }

    /**
     * Renvoie la quantité collectée par l'agent et la réinitialise.
     *
     * @return la quantité collectée
     */
    public int videCollecte() {
        int qte = qteCollectee;
        qteCollectee = 0;
        return qte;
    }
}
```

```java
/**
 * Exception qui indique qu'il n'y a pas eu de collecte avant
 * l'appel à une méthode qui utilise les collectes.
 */

public class PasDeCollecteException extends Exception {
```

```java
    /**
     * @param methodeActuelle la méthode dans laquelle l'erreur s'est produite
     * @param methodeAAppeler la méthode à appeler avant la méthode actuelle
pour éviter l'erreur
     */
    public PasDeCollecteException(String methodeActuelle, String
methodeAAppeler) {
        super("Pas de collecte ! Appelez la méthode " + methodeAAppeler + "()
avant d'appeler " + methodeActuelle + "().");
    }
}
```

```java
/**
 *
 * Représentation des gisements de pétrole présents sur le terrain.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 *
 */

public class Petrole extends Ressource {
    /**
     * Constructeur qui initialise la quantité de pétrole
     * @param quantite quantité de pétrole
     */
    public Petrole(int quantite) {
        super("Pétrole", quantite);
    }
}
```

```java
/**
 * Classe mère des plastiques polluant et biodégradable.
 * Classes filles sont {@link PlastiquePolluant} et {@link
PlastiqueBioDegradable}.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public abstract class Plastique extends Ressource {
    private int age;

    /**
     * Constructeur qui initialise le type du plastique et sa quantité. Au
moment de création, l'âge est 0 par défaut.
     *
     * @param type type du plastique
     * @param qte  quantité du plastique
     */
    public Plastique(String type, int qte) {
        super(type, qte);
        age = 1; // à l'instant du dépôt ou de création
    }

    /**
     * Permet d'avoir accès à l'âge du plastique.
     *
     * @return l'âge du plastique
     */
    public int getAge() {
```

```
        return age;
    }

    /**
     * Augmente l'âge du plastique d'un an
     */
    public void augmenteAge() {
        age++;
    }

    /**
     * Renvoie les infos relatives au plastique, notamment l'âge.
     *
     * @return les infos relatives au plastique, notamment l'âge.
     */
    @Override
    public String toString() {
        return super.toString() + " age: " + getAge();
    }
}
```

```
/**
 * Plastique biodégradable qui se décompose dans l'eau après
 * un bout de temps, contrairement au {@link PlastiquePolluant}
 * qui a besoin d'être ramassé par un {@link TravailleurUsine}.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class PlastiqueBioDegradable extends Plastique {
    private int dureeDeVie;

    /**
     * Constructeur qui initialise la quantité du plastique biodégradable ainsi
que sa durée de vie
     *
     * @param qte        quantité du plastique biodégradable
     * @param dureeDeVie durée de vie, l'âge auquel se décompose le plastique
     */
    public PlastiqueBioDegradable(int qte, int dureeDeVie) {
        super("PBD", qte);

        this.dureeDeVie = dureeDeVie;
    }

    /**
     * Constructeur qui initialise la quantité du plastique biodégradable ainsi
que sa durée de vie
     *
     * @param pp un plastique polluant
     */
    public PlastiqueBioDegradable(PlastiquePolluant pp) {
        super("PBD", pp.getQuantite());

        this.dureeDeVie = Simulation.randEntre(1, 2);
    }

    /**
     * Vérifie si assez de temps est passé pour que le plastique biodégradable
se décompose
     *
```

```
     * @return booléen qui indique si la décomposition est possible ou pas
     */
    public boolean estDecompositionPossible() {
        return getAge() >= dureeDeVie;
    }

    /**
     * Renvoie des informations sur le plastique biodégradable
     *
     * @return l'ID et la quantité du plastique biodégradable, s'il est présent
     * sur le terrain et ses coordonnées si oui, et si le plastique s'est
décomposé ou non
     */
    @Override
    public String toString() {
        return super.toString() +
                ", dureeDeVie: " + dureeDeVie + "." +
                ((estDecompositionPossible()) ? " S'est décomposé !" : " Besoin
plus de temps pour la décomposition...");
    }
}
```

```
/**
 * Plastique polluant qui ne se décompose pas dans l'eau
 * tout seul. Il est ramassé par les {@link TravailleurUsine}
 * pour être recyclé en {@link PlastiqueBioDegradable}.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class PlastiquePolluant extends Plastique {
    private int ageLimiteDeRecyclage;

    /**
     * Constructeur qui initialise la quantité du plastique polluant
     *
     * @param qte                   quantité du plastique polluant
     * @param ageLimiteDeRecyclage âge après lequel le plastique ne peut plus
être recyclé
     */
    public PlastiquePolluant(int qte, int ageLimiteDeRecyclage) {
        super("PP", qte);

        this.ageLimiteDeRecyclage = ageLimiteDeRecyclage;
    }

    /**
     * Recylce le plastique polluant en le transformant en plastique
biodégradable
     *
     * @return le nouveau plastique biodégradable
     */
    public boolean estRecyclagePossible() {
        return getAge() < ageLimiteDeRecyclage;
    }

    /**
     * Renvoie des informations sur le plastique polluant
     *
     * @return l'ID, la quantité du plastique polluant, et s'il a été recyclé
     */
```

```
    @Override
    public String toString() {
        return super.toString() +
                ", ageLimiteDeRecyclage: " + ageLimiteDeRecyclage + "." +
                ((estRecyclagePossible()) ? " Recyclage encore possible !" : "
Ne peux plus être recyclé");
    }
}
```

```
import java.util.ArrayList;

/**
 * Simulation du terrain. Elle contient et modifie plusieurs transformations :
 * <ul>
 *  <li>la création d'une liste de {@link TechnicienPetrolier}</li>
 *  <li>la création d'une liste de {@link TravailleurUsine}</li>
 *  <li>la creation de {@link Petrole}</li>
 *  <li>la création du {@link PlastiquePolluant} à partir du pétrole,</li>
 *  <li>le recyclage du plastique polluant en {@link PlastiqueBioDegradable},
et</li>
 *  <li>la décomposition du plastique biodégradable jeté dans le terrain.</li>
 * </ul>
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class Simulation {
    private static Simulation INSTANCE;

    public static Simulation getInstance() {
        if (INSTANCE == null) INSTANCE = new Simulation();
        return INSTANCE;
    }

    private final int[] SIZE_TERRAIN = {10, 10};
    private final Terrain terrain;
    private final int nbPetrole;
    private final int nbTPs;
    private final int nbTUs;

    private final ArrayList<TechnicienPetrolier> techniciens;
    private int totalExtraction;

    private final ArrayList<TravailleurUsine> travailleurs;
    private ArrayList<PlastiquePolluant> plastiqueRamasse;

    private final ArrayList<PlastiquePolluant> pps;

    private final ArrayList<PlastiqueBioDegradable> pbds;
    private final ArrayList<PlastiqueBioDegradable> pbdsATraiter;

    private Simulation() {
        this.terrain = new Terrain(SIZE_TERRAIN[0], SIZE_TERRAIN[1]);
        nbPetrole = randEntre(3, 5);
        nbTPs = randEntre(3, 4);
        nbTUs = randEntre(5, 7);

        techniciens = new ArrayList<TechnicienPetrolier>();
        totalExtraction = 0;

        travailleurs = new ArrayList<TravailleurUsine>();
```

```
        plastiqueRamasse = new ArrayList<PlastiquePolluant>();

        pps = new ArrayList<PlastiquePolluant>();

        pbds = new ArrayList<PlastiqueBioDegradable>();
        pbdsATraiter = new ArrayList<PlastiqueBioDegradable>();
    }

    /**
     * Constructeur qui copie une simulation existante à un moment donné (pour
des statistiques et études potentielles)
     */
    private Simulation(Simulation sim) {
        terrain = new Terrain(sim.terrain.nbLignes, sim.terrain.nbColonnes);
        nbPetrole = sim.nbPetrole;
        nbTPs = sim.nbTPs;
        nbTUs = sim.nbTUs;

        techniciens = new ArrayList<TechnicienPetrolier>(sim.techniciens);
        totalExtraction = sim.totalExtraction;

        travailleurs = new ArrayList<TravailleurUsine>(sim.travailleurs);
        plastiqueRamasse = new
ArrayList<PlastiquePolluant>(sim.plastiqueRamasse);

        pps = new ArrayList<PlastiquePolluant>(sim.pps);

        pbds = new ArrayList<PlastiqueBioDegradable>(sim.pbds);
        pbdsATraiter = new ArrayList<PlastiqueBioDegradable>(sim.pbdsATraiter);
    }

    // Méthodes terrain

    /**
     * Affiche le terrain.
     *
     * @param nbCaracteres le nombre de caractères que doit afficher chaque case
     */
    public void afficheTerrain(int nbCaracteres) {
        terrain.affiche(nbCaracteres);
//        System.out.println("Informations sur le terrain:\n" + terrain + "\n");
    }

    /**
     * Affiche le terrain avec 7 caractères.
     */
    public void afficheTerrain() {
        afficheTerrain(7);
    }

    /**
     * Renvoie le nombre de cases remplies sur le terrain.
     *
     * @return le nombre de cases remplies sur le terrain
     */
    public int getCasesRemplies() {
        return terrain.lesRessources().size();
    }

    /**
     * Crée et affiche (si demandé) les Techniciens Pétroliers.
     *
     * @param affiche si la méthode doit afficher les techniciens
     */
```

```
    public void createTechniciens(boolean affiche) {
        if (affiche) System.out.println("Nos Techniciens Pétroliers :");

        for (int i = 0; i < nbTPs; i++) {
            int capaciteDeCollecte = randEntre(75, 150);
            int capaciteDeBaril = randEntre(30, 50);
            int nbBarils = randEntre(5, 10);
            techniciens.add(new TechnicienPetrolier(capaciteDeCollecte,
capaciteDeBaril, nbBarils, terrain));
            if (affiche) System.out.println(techniciens.get(i));
        }

        if (affiche) System.out.println();
    }

    /**
     * Affiche les Techniciens Pétroliers.
     */
    public void afficheTechniciens() {
        for (TechnicienPetrolier tp : techniciens) {
            System.out.println(tp);
        }
    }

    /**
     * Fait parcourir le terrain aux Techniciens Pétroliers,
     * qui extraient du pétrole s'ils en trouvent.
     */
    public void runTechniciens() {
        for (TechnicienPetrolier technicien : techniciens) {
            System.out.println("Le Technicien numéro " + technicien.ident + "
parcourt le terrain.");
            terrainParcours:
            for (int i = technicien.getPosX(); i < SIZE_TERRAIN[0] &&
!technicien.estPlein(); i++) {
                technicien.seDeplacer(i, 0);
                for (int j = technicien.getPosY(); j < SIZE_TERRAIN[1] &&
!technicien.estPlein(); j++) {
                    if (!estDeplacementPossible(technicien, i, j)) {
                        System.out.println("Oops, un autre agent se trouve sur
(" + i + "," + j + "), je saute cette case.");
                        continue;
                    }

                    technicien.seDeplacer(i, j);
                    if (terrain.getCase(i, j) instanceof Petrole) {
                        Petrole p = (Petrole) terrain.getCase(i, j);
                        System.out.println("J'ai trouvé " + p.getQuantite() + "L
de pétrole en (" + p.getX() + ", "
                                + p.getY() + ")");
                    }
                    try {
                        int collecte = technicien.collecter();
                        if (collecte != -1) {
                            System.out.println("Je viens de collecter " +
collecte + "L de pétrole. Mes barils contiennent " +
technicien.getQuantiteCollectee() + "L en tout.");
                            break terrainParcours;
                        }
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                }
            }
```

```
            afficheTerrain();
        }
    }

    /**
     * Vérifie si la case où se déplace l'agent est vide ou pas.
     *
     * @param agent l'agent en question
     * @param x     coordonnées x de la case
     * @param y     coordonnées y de la case
     * @return si la case où se déplace l'agent est vide
     */
    private boolean estDeplacementPossible(Agent agent, int x, int y) {
        for (TechnicienPetrolier tp : techniciens) {
            if (tp.ident != agent.ident && tp.getPosX() == x && tp.getPosY() ==
y) return false;
        }
        for (TravailleurUsine tu : travailleurs) {
            if (tu.ident != agent.ident && tu.getPosX() == x && tu.getPosY() ==
y) return false;
        }
        return true;
    }

    /**
     * Vide la liste de techniciens.
     */
    public void removeTechniciens() {
        techniciens.clear();
    }

    /**
     * Ajouter le pétrole au terrain.
     */
    public void createPetrole() {
        for (int i = 0; i < nbPetrole; i++) {
            int x = randEntre(0, 4);
            int y = randEntre(5, 9);
            int qte = randEntre(200, 300); // En litres
            Petrole petrole = new Petrole(qte);
            petrole.setPosition(x, y);
            terrain.setCase(x, y, petrole);
        }
    }

    /**
     * Vide la totalité du pétrole stocké dans les barils des techniciens.
     *
     * @return le volume de pétrole total extrait par les techniciens
     * @throws PasDeCollecteExceptionon s'il n'y a pas de collectes
     */
    public int videExtractionPetrole() throws PasDeCollecteException {
        int qteExtraction = 0;
        for (TechnicienPetrolier tp : techniciens) {
            qteExtraction += tp.videCollecte();
        }
        if (qteExtraction == 0) {
            throw new PasDeCollecteException("videExtractionPetrole",
"runTechniciens");
        }
        totalExtraction += qteExtraction;
        return qteExtraction;
    }
```

```java
    /**
     * Produit du plastique polluant à partir du pétrole total extrait.
     *
     * @throws PasDeCollecteExceptions'il n'y a pas de collectes
     */
    public void produirePlastique() throws PasDeCollecteException {
        if (totalExtraction == 0) {
            throw new PasDeCollecteException("produirePlastique",
"videExtractionPetrole");
        }
        int nbPlastiqueProduit = (int) (totalExtraction / 10.0);
        for (int i = 0; i < nbPlastiqueProduit; i++) {
            int randQte = randEntre(3, 5);
            int ageLimiteDeRecyclage = randEntre(3, 6);
            pps.add(new PlastiquePolluant(randQte, ageLimiteDeRecyclage));
        }
        totalExtraction = 0;
    }

    /**
     * Pollue le terrain avec des plastiques polluants.
     *
     * @param affiche si la méthode doit afficher les éléments
     */
    public void polluer(boolean affiche) {
        for (PlastiquePolluant pp : pps) {
            int x = randEntre(5, 9);
            int y = randEntre(0, 4);
            pp.setPosition(x, y);
            terrain.setCase(x, y, pp);
            if (affiche) System.out.println(pp);
        }
        pps.clear();
    }

    /**
     * Crée et affiche (si demandé) les Travailleurs d'usine.
     *
     * @param affiche si la méthode doit afficher les travailleurs
     */
    public void createTravailleurs(boolean affiche) {
        if (affiche) System.out.println("Nos Travailleurs à l'usine :");

        for (int i = 0; i < nbTUs; i++) {
            int capaciteDeCollecte = randEntre(5, 8);
            int capaciteDeStockage = randEntre(7, 13);
            travailleurs.add(new TravailleurUsine(capaciteDeCollecte,
capaciteDeStockage, terrain));
            if (affiche) System.out.println(travailleurs.get(i));
        }

        if (affiche) System.out.println();
    }

    /**
     * Fait parcourir le terrain aux Travailleurs d'usine,
     * qui collectent du plastique polluant s'ils en trouvent.
     */
    public void runTravailleurs() {
        for (TravailleurUsine travailleur : travailleurs) {
            System.out.println("Le Travailleur numéro " + travailleur.ident + "
parcourt le terrain.");
            terrainParcours:
```

```
            for (int i = travailleur.getPosX(); i < SIZE_TERRAIN[0] &&
!travailleur.estPlein(); i++) {
                travailleur.seDeplacer(i, 0);
                for (int j = travailleur.getPosY(); j < SIZE_TERRAIN[1] &&
!travailleur.estPlein(); j++) {
                    if (!estDeplacementPossible(travailleur, i, j)) {
                        System.out.println("Oops, un autre agent se trouve sur
(" + i + "," + j + "), je saute cette case.");
                        continue;
                    }

                    travailleur.seDeplacer(i, j);
                    if (terrain.getCase(i, j) instanceof PlastiquePolluant) {
                        PlastiquePolluant tPP = (PlastiquePolluant)
terrain.getCase(i, j);
                        System.out.println("J'ai trouvé " + tPP.getQuantite() +
"kg de plastique en (" + tPP.getX()
                                + ", " + tPP.getY() + ")");
                    }
                    try {
                        int collecte = travailleur.collecter();
                        if (collecte != -1) {
                            System.out.println("Je viens de collecter " +
collecte + "kg de plastique. J'ai " + travailleur.getQuantiteCollectee() + "kg
en tout.");
                            break terrainParcours;
                        }
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                }
            }

            afficheTerrain();
        }
    }

    /**
     * Vide la liste de travailleurs.
     */
    public void removeTravailleurs() {
        travailleurs.clear();
    }

    /**
     * Vide la totalité du plastique polluant stocké avec les travailleurs.
     *
     * @return le volume de plastique polluant total collecté par les
travailleurs
     * @throws PasDeCollecteException s'il n'y a pas de collectes
     */

    public int videRamassagePlastique() throws PasDeCollecteException {
        plastiqueRamasse.clear();

        for (TravailleurUsine tu : travailleurs) {
            plastiqueRamasse.addAll(tu.getCollectes());
            tu.videCollecte();
        }

        if (plastiqueRamasse.isEmpty()) {
            throw new PasDeCollecteException("videRamassagePlastique",
"runTravailleurs");
        }
```

```
        return plastiqueRamasse.size();
    }

    /**
     * Produit du plastique biodégradable à partir du plastique polluant total
ramassé.
     *
     * @return plastiques biodégradables recyclés
     * @throws PasDeCollecteException s'il n'y a pas de collectes
     */
    public int recyclerTout() throws PasDeCollecteException {
        if (plastiqueRamasse.size() == 0) {
            throw new PasDeCollecteException("recyclerTout",
"videRamassagePlastique");
        }

        for (PlastiquePolluant pp : plastiqueRamasse) {
            pbdsATraiter.add(new PlastiqueBioDegradable(pp));
        }

        plastiqueRamasse.clear();

        return pbdsATraiter.size();
    }

    /**
     * Rejette le plastique bio dans le terrain pour qu'il se décompose
     *
     * @param affiche si la méthode doit afficher les éléments
     */
    public void rejettePlastiquesBio(boolean affiche) {
        for (PlastiqueBioDegradable pbd : pbdsATraiter) {
            int x = randEntre(0, 4);
            int y = randEntre(0, 4);
            if (terrain.caseEstVide(x, y)) {
                pbds.add(pbd);
                pbd.setPosition(x, y);
                terrain.setCase(x, y, pbd);
//                if (affiche) System.out.println(pbd);
            }
        }
        pbdsATraiter.clear();
    }

    /**
     * Essaye de faire décomposer le plastique biodégradable.
     *
     * @return la quantité de plastique biodégradable décomposé
     */
    public int decomposerTout() {
        int qteDecomposee = 0;
        ArrayList<PlastiqueBioDegradable> aEnlever = new
ArrayList<PlastiqueBioDegradable>();
        for (PlastiqueBioDegradable pbd : pbds) {
            if (pbd.estDecompositionPossible()) {
                terrain.videCase(pbd.getX(), pbd.getY());
                aEnlever.add(pbd);
                qteDecomposee++;
            }
        }
        pbds.removeAll(aEnlever);
        return qteDecomposee;
    }
```

```java
    /**
     * Augmente l'âge de toutes les ressources.
     */
    public void augmenteAge() {
        ArrayList<Ressource> toutesRessources = new ArrayList<Ressource>();
        toutesRessources.addAll(pps);
        toutesRessources.addAll(pbds);
        for (Ressource r : toutesRessources) {
            if (r instanceof Plastique) ((Plastique) r).augmenteAge();
        }
    }

    /**
     * Renvoie une copie de Simulation dans son état actuel
     * (pour des statistiques et études potentielles).
     *
     * @return une copie de Simulation dans son état actuel
     */
    public Simulation saveState() {
        return new Simulation(this);
    }

    /**
     * @param min la borne inférieure
     * @param max la borne supérieure
     * @return un nombre entier appartenant à [min, max]
     */
    public static int randEntre(int min, int max) {
        return (int) (Math.random() * Math.abs(max + 1 - min) + min);
    }

    @Override
    public String toString() {
        return "Simulation d'un terrain de " + SIZE_TERRAIN[0] + "x" +
SIZE_TERRAIN[1] +
                " avec " + nbPetrole + " cases de pétrole et " + nbTPs + "
travailleurs.";
    }
}
```

```java
/**
 * Représentation d'un Technicien Pétrolier, agent qui parcourt le terrain
 * et collecte du pétrole dans des barils lorsqu'il en trouve. Il peut aussi
 * transformer ses réserves en plastique.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class TechnicienPetrolier extends Collecteur {
    private int nbBarils;
    /**
     * Constructeur qui initialise la capacité de collecte et de stockage du
TechnicienPetrolier
     *
     * @param capaciteDeCollecte capacité de pétrole que peut collecter le
TechnicienPetrolier
     * @param capaciteDeBaril    la capacité des barils
     * @param nbBarils           le nombre de barils
     * @param t                  terrain sur lequel se trouve le
TechnicienPetrolier
```

```
     */
    public TechnicienPetrolier(int capaciteDeCollecte, int capaciteDeBaril, int
nbBarils, Terrain t) {
        super("TechnicienPetrolier", t, capaciteDeCollecte, capaciteDeBaril *
nbBarils);

        this.nbBarils = nbBarils;
    }

    /**
     * Permet de collecter le Pétrole qui peut se trouver dans la même case du
TechnicienPetrolier,
     * si la capacité de collecte et la capacité de stockage le permettent.
     *
     * @throws Exception s'il n'y a plus de place pour stocker le pétrole
     */
    @Override
    public boolean verifierCollectePossible() throws Exception {
        if (estPlein()) throw new Exception("Je ne peux pas stocker plus de
pétrole avec moi.");

        Ressource ressourceACollecter = getTerrain().getCase(getPosX(),
getPosY());

        // S'il n'y a pas de ressource sur cette case ou la ressource n'est pas
du pétrole,
        // on retourne false.
        return ressourceACollecter instanceof Petrole;
    }

    /**
     * Renvoie des informations sur le TechnicienPetrolier
     *
     * @return l'ID et la position du TechnicienPetrolier, le nombre de barils
et leur capacité de stockage,
     * la capacité de collecte et la quantité collectée
     */
    @Override
    public String toString() {
        return super.toString()
                + " J'ai " + nbBarils + " barils qui, réunis, peuvent contenir "
+ capaciteDeStockage + "L de pétrole."
                + " À chaque collecte, je peux extraire " + capaciteDeCollecte +
"L seulement et j'en ai déjà " + getQuantiteCollectee() + "L.";
    }
}
```

```
/**
 * Test de la Simulation de l'écosystème créé.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class TestSimulation {
    public static void main(String[] args) {
        // Instance de classe Simulation
        Simulation simulation = Simulation.getInstance();

        // Ajouter le pétrole au terrain
        simulation.createPetrole();
```

```
        // Affiche terrain
        simulation.afficheTerrain(7);

        // Crée et affiche les Techniciens Pétroliers
        simulation.createTechniciens(true);

        // Crée et affiche les Travailleurs d'usine
        simulation.createTravailleurs(true);

        int i = 1;
        while (simulation.getCasesRemplies() != 0) {
            System.out.println("|-------------------------------------------------
----|");
            System.out.println("|--------------------  Cycle " + i + "  --------
------------|");
            System.out.println("|-------------------------------------------------
----|");

            // Parcourir et extraire le pétrole
            simulation.runTechniciens();

            try {
                // Vider les extractions de pétrole, produire du plastique et
polluer le terrain
                System.out.println("Nos Techniciens Pétroliers après extraction
:");
                simulation.afficheTechniciens();
                int totalExtraction = simulation.videExtractionPetrole();
                System.out.println("Ils ont collecté " + totalExtraction + "L en
tout.\n");
                System.out.println("L'usine produit du plastique...");
                simulation.produirePlastique();
                simulation.polluer(false);
                System.out.println("Du plastique polluant a été jeté dans
l'eau\n");
            } catch (PasDeCollecteException e) {
                System.out.println(e.getMessage());
            }

            // Affiche terrain
            simulation.afficheTerrain(7);
            // Parcourir et collecter le plastique polluant
            simulation.runTravailleurs();

            try {
                // Vider les ramassages de plastique, tout recycler et les
laisser se décomposer dans le terrain
                System.out.println("Nos Travailleurs à l'usine après ramassage
:");
                int totalRamassage = simulation.videRamassagePlastique();
                System.out.println("Ils ont collecté " + totalRamassage + "kg de
plastique polluant en tout.\n");
                System.out.println("Recyclage en cours...");
                int nbPlastiquesBio = simulation.recyclerTout();
                simulation.rejettePlastiquesBio(true);
                System.out.println(nbPlastiquesBio + " plastiques biodégradables
ont été jetés dans l'eau\n");
            } catch (PasDeCollecteException e) {
                System.out.println(e.getMessage());
            }

            // Affiche terrain
            simulation.afficheTerrain(7);
            // Essaye de décomposer le plastique biodégradable jeté dans l'eau
```

```
            int qteDecomposee = simulation.decomposerTout();
            System.out.println(qteDecomposee + " plastiques biodégradables se
sont décomposés \n");
            // Affiche terrain
            simulation.afficheTerrain(7);

            simulation.augmenteAge();

            // Ici, on peut faire une copie de la simulation pour enregistrer
            // l'état actuel et l'étudier par la suite. On utilise :
            //
            // simulation.saveState();
            //
            // ou, si on est hors du main, ou dans une autre classe, on peut
utiliser :
            //
            // Simulation.getInstance().saveState();

            i++;
        }

        System.out.println("L'écosystème s'est éteint au bout de " + i + "
cycles");
    }
}
```

```
import java.util.ArrayList;

/**
 * Représentation d'un Travailleur à l'usine, agent qui parcourt le terrain,
 * ramasse et stocke du plastique polluant lorsqu'il en trouve. Il peut aussi
 * transformer ses réserves en plastique bio-dégradable.
 *
 * @author Toufic BATACHE (LU2IN002 2022dec)
 * @author Haya MAMLOUK (LU2IN002 2022dec)
 */

public class TravailleurUsine extends Collecteur {
    private final ArrayList<PlastiquePolluant> collectes;

    /**
     * Constructeur qui initialise la capacité de collecte et de stockage du
TravailleurUsine
     *
     * @param capaciteDeCollecte capacité de collecte du TravailleurUsine
     * @param capaciteDeStockage capacité de stockage du TravailleurUsine
     * @param t                  terrain sur lequel se trouve le
TravailleurUsine
     */
    public TravailleurUsine(int capaciteDeCollecte, int capaciteDeStockage,
Terrain t) {
        super("TravailleurUsine", t, capaciteDeCollecte, capaciteDeStockage);

        this.collectes = new ArrayList<PlastiquePolluant>();
    }

    /**
     * Permet de collecter la ressource qui peut se trouver
     * dans la même case que le travailleur d'usine.
     *
     * @return le volume de la collecte
     * @throws Exception si la collecte n'est pas possible pour une raison
```

```
fatale
     */
    @Override
    public int collecter() throws Exception {
        int nbCollecte = super.collecter();
        if (nbCollecte != -1) {
            collectes.add(new PlastiquePolluant(nbCollecte, -1));
        }
        return nbCollecte;
    }

    /**
     * Renvoie les collectes de plastique polluant.
     *
     * @return les collectes de plastique polluant
     */
    public ArrayList<PlastiquePolluant> getCollectes() {
        return collectes;
    }

    /**
     * Permet de collecter le PlastiquePolluant qui peut se trouver dans la même
case du TravailleurUsine,
     * si la capacité de collecte et la capacité de stockage le permettent.
     *
     * @throws Exception s'il n'y a plus de place pour stocker le plastique
     */
    @Override
    public boolean verifierCollectePossible() throws Exception {
        if (estPlein())
            throw new Exception("Je ne peux pas stocker plus de plastique avec
moi. Besoin de déposer à l'usine.");

        Ressource ressourceACollecter = getTerrain().getCase(getPosX(),
getPosY());

        // S'il n'y a pas de ressource sur cette case ou la ressource n'est pas
du pétrole, ou
        // si le recyclage n'est pas possible, on retourne false.
        return ressourceACollecter instanceof PlastiquePolluant &&
                ((PlastiquePolluant)
ressourceACollecter).estRecyclagePossible();
    }

    @Override
    public int videCollecte() {
        int nbCollecte = super.videCollecte();
        collectes.clear();
        return nbCollecte;
    }

    /**
     * Renvoie des informations sur le TravailleurUsine.
     *
     * @return l'ID et la position du TravailleurUsine, sa capacité de stockage,
     * sa capacité de collecte et la quantité collectée
     */
    @Override
    public String toString() {
        return super.toString() +
                " Je peux stocker " + capaciteDeStockage + "kg de plastique avec
moi." +
                " À chaque collecte, je peux ramasser " + capaciteDeCollecte +
"kg de plastique polluant" +
```

```
                " et j'en ai déjà " + getQuantiteCollectee() + "kg sur moi.";
    }
}
```