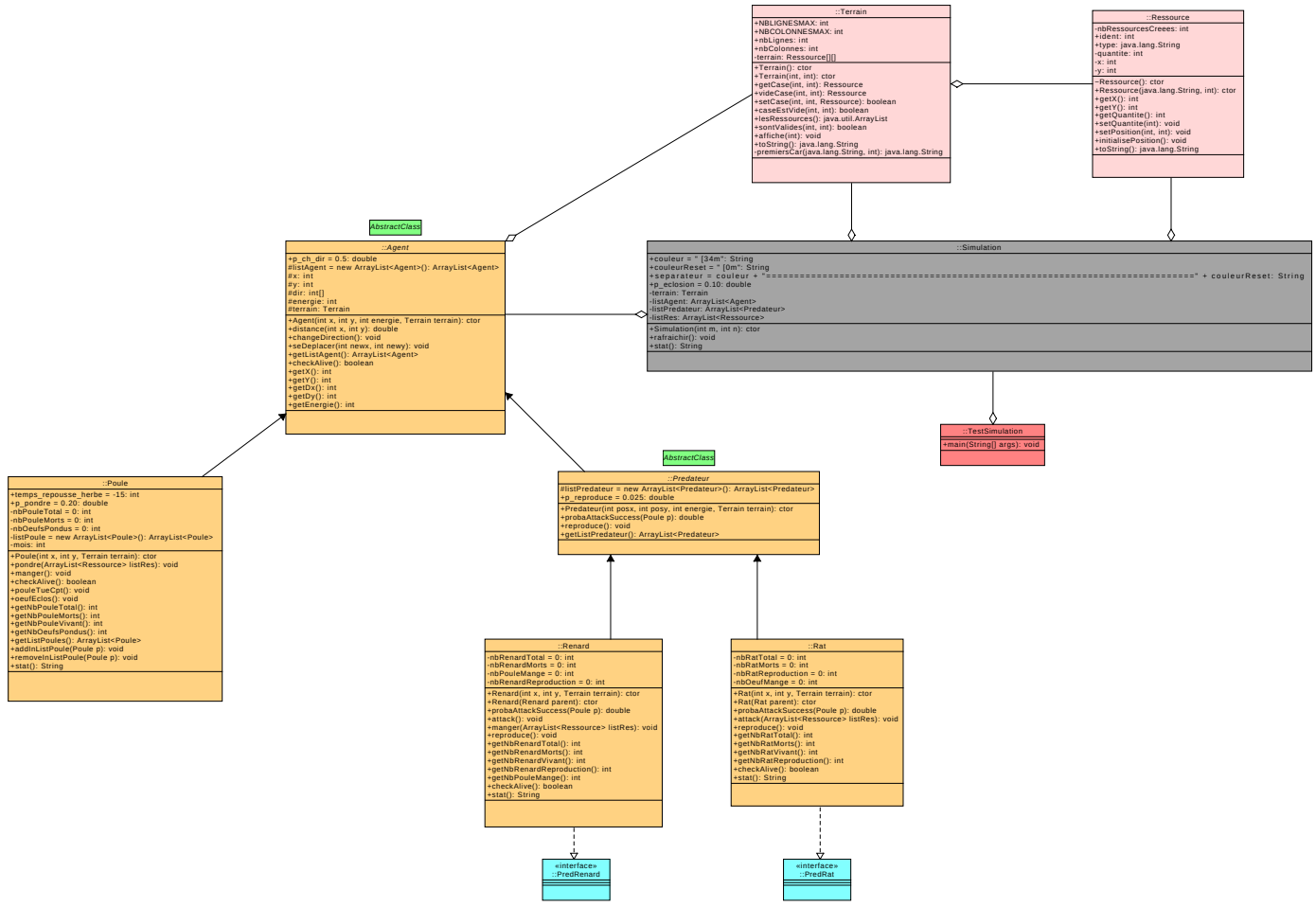
Projet LU2IN002 - 2022-2023

Numéro du groupe de TD/TME : Groupe 4

Nom : PINHO FERNANDES	*Nom :* DURBIN
Prénom : Enzo	*Prénom :* Deniz Ali
N° étudiant : 21107465	*N° étudiant :* 21111116

Thème choisi (en 2 lignes max.)
Simulation de la problématique entre les poules et leurs ennemis/prédateurs. Les poules mangent l'herbe et pondent les oeufs, les renards mangent les poules, les rats mangent les oeufs.

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	Agent, Poule, Renard, Rat
Classe avec membres et méthodes statiques	Agent, Predateur, Poule, Renard, Rat, Simulation

Classe abstraite et méthode abstraite	Agent, Predateur
Interface	ProdRenard, ProdRat
Classe avec un constructeur par copie ou clone()	Renard, Rat (Pour la reproduction)
Définition de classe étendant Exception	
Gestion des exceptions	TestSimulation

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

Notre projet tente de simuler les attaques que peuvent subir les poules de la part des renards et des rats. En effet, c'est un problème très récurrent. De plus, les rats attaquent les oeufs des poules. Ces trois là sont donc les agents de notre projet. Il y a tout un système d'énergie, de reproduction, de probabilité… afin de faire une simulation intéressante d'après nous.

Les 3 ressources sont :

→ L'herbe qui pousse sans l'aide des agents
→ Les oeufs qui seront pondus par les poules et écloeront à un moment.
→ La viande, elle sera déposé sur le terrain si une poule meurt d'elle-même.

```
import java.util.ArrayList;

public abstract class Agent {
    /*
     * Cette classe abstraite représente les agents en général.
     * Elle possédera la liste de tous les agents crées et actifs de la simulation, ainsi que la probabilité de changement de direction.
     * Chaque agent aura une position x et y, une direction, une énergie et le terrain associé à eux.
     */


    public static final double p_ch_dir = 0.5;
    public static ArrayList<Agent> listAgent = new ArrayList<Agent>();

    protected int x;
    protected int y;
    protected int[] dir;
    protected int energie;
    protected Terrain terrain;

    public Agent(int x, int y, int energie, Terrain terrain){
        this.x = x;
        this.y = y;

        this.dir = new int[2];
        dir[0] = (int)(Math.random() * 3 - 1);
        dir[1] = (int)(Math.random() * 3 - 1);

        this.energie = energie;
        this.terrain = terrain;

        listAgent.add(this);
    }



    /*
     * Permet de calculer la distance euclidienne entre le point (x,y) et l'agent.
     */
    public double distance(int x, int y){
        return Math.sqrt(Math.pow(this.x - x, 2) + Math.pow(this.y - y, 2));
    }



    /*
     * Fonction de changement de direction d'un agent.
     * Si le hasard l'accepte, alors la nouvelle direction [dx,dy] de l'agent sera choise aléatoirement entre -1 et 1 compris.
     */
    public void changeDirection(){
        if (Math.random() < p_ch_dir){
            dir[0] = (int)(Math.random() * 3 - 1);
```

```
            dir[1] = (int)(Math.random() * 3 - 1);
        }
    }



    /*
     * Fonction permettant de se déplacer.
     * Chaque agent perd un point d'énergie par déplacement.
     * Si un autre agent occupe la position convoitée, il restera à sa place.
     */
    public void seDeplacer(int newx, int newy){
        this.energie--;
        int posx = (newx + terrain.nbLignes) % (terrain.nbLignes);
        int posy = (newy + terrain.nbColonnes) % (terrain.nbColonnes);

        for (Agent agent : listAgent)
            if (agent.x == posx && agent.y == posy)
                return;

        this.x = posx; this.y = posy;
    }


    public static ArrayList<Agent> getListAgent() { return listAgent; }


    /*
     * Fonction qui permet de supprimer un agent si son énergie tombe à 0.
     */
    public abstract boolean checkAlive();

    public int getX() { return x; }
    public int getY() { return y; }
    public int getDx() { return dir[0]; }
    public int getDy() { return dir[1]; }
    public int getEnergie() { return energie; }
}
```

```
import java.util.ArrayList;

public class Poule extends Agent{
    /*
     * Cette classe représente les poules, qui sont des agents.
     * Elle contient les constantes du temps de la repousse de l'herbe après avoir été mangée, ainsi que la probabilité qu'elle ponde un oeuf.
     * Elle contiendra aussi une liste de toutes les poules crées et actives de la simulation.
     */
    public static final int temps_repousse_herbe = -15;
    public static final double p_pondre = 0.20;

    // Statistiques.
    private static int nbPouleTotal = 0;
    private static int nbPouleMorts = 0;
    private static int nbOeufsPondus = 0;

    private static ArrayList<Poule> listPoule = new ArrayList<Poule>();

    private int mois;

    public Poule(int x, int y, Terrain terrain){
        super(x, y, (int)(Math.random()*5 + 10), terrain);
        this.mois = (int)(Math.random()*6);
        nbPouleTotal++;

        listPoule.add(this);
    }



    /*
     * Fonction qui permet de faire pondre une poule.
     * Si la poule à l'âge requis (6 mois), et que le hasard le veut bien, alors elle pondra un oeuf.
     * L'oeuf sera rajouté à la liste de ressources et placé sur le terrain.
     * S'il est déjà dans le terrain, alors on augmente la quantité de 1.
     * Si la position est occupée, on suppose que la poule fait le ménage pour son enfant.
     */
    public void pondre(ArrayList<Ressource> listRes){
        if (mois >= 6 && Math.random() < p_pondre){
            if (terrain.caseEstVide(x, y)){
                Ressource oeuf = new Ressource("Oeufs", 1);
                terrain.setCase(x, y, oeuf);
                listRes.add(oeuf);
                nbOeufsPondus++;
                System.out.println("Un nouvel oeuf a été pondu en (" + x + "," + y + " !");
                return;
            }

            Ressource res = terrain.getCase(x,y);

            if (res.type.equals("Oeufs"))
```

```
            res.setQuantite(res.getQuantite() + 1);
        else{
            terrain.videCase(x, y);
            listRes.remove(res);
            Ressource oeuf = new Ressource("Oeufs", 1);
            terrain.setCase(x, y, oeuf);
            listRes.add(oeuf);
        }

        System.out.println("Un nouvel oeuf a été pondu en (" + x + "," + y + ") !");
        nbOeufsPondus++;
    }

    mois++;
}



/*
 * Fonction qui permet à la poule de manger l'herbe.
 * On rajoute la quantité d'herbe à ses points d'énergie.
 * L'herbe prendre <temps_repousse_herbe> itérations avant d'être mangeable de nouveau.
 */
public void manger(){
    if (!terrain.caseEstVide(x, y)){
        Ressource res = terrain.getCase(x, y);
        if (res.type.equals("  __  ") && res.getQuantite() > 0){
            this.energie += res.getQuantite() / 3;
            res.setQuantite(temps_repousse_herbe);
        }
    }
}


public boolean checkAlive(){
    if (energie <= 0){
        listPoule.remove(this);
        listAgent.remove(this);
        nbPouleMorts++;
        System.out.println("Une poule est morte d'épuisement en (" + x + "," + y + ") !");
        return false;
    }

    return true;
}



/*
 * Fonctions utilitaires pour les statistiques, l'ajout dans la liste des Poules...
 */
public static void pouleTueCpt() { nbPouleMorts++; }
```

```
    public static void oeufEclos() { nbPouleTotal++; }

    public static int getNbPouleTotal() { return nbPouleTotal; }
    public static int getNbPouleMorts() { return nbPouleMorts; }
    public static int getNbPouleVivant() { return nbPouleTotal - nbPouleMorts; }
    public static int getNbOeufsPondus() { return nbOeufsPondus; }

    public static ArrayList<Poule> getListPoules() { return listPoule; }

    public static void addInListPoule(Poule p) { listPoule.add(p); }
    public static void removeInListPoule(Poule p) { listPoule.remove(p); }




    /*
     * Fonction qui permet de récupérer les statistiques de fin de simulation des poules.
     */
    public static String stat(){
        String message = "Nombre de poules au total : " + nbPouleTotal;
        message += "\nNombre de poules mortes : " + nbPouleMorts;
        message += "\nNombre d'oeufs pondus : " + nbOeufsPondus;

        return message;
    }
}
```

```
import java.util.ArrayList;

public abstract class Predateur extends Agent{
    /*
     * C'est la classe abstract des Prédateus (Renard et Rat).
     * Elle contiendra la liste de tous les Prédateurs crées et actifs de la simulation, ainsi que la contante de la probabilité de reproduction.
     */

    protected static ArrayList<Predateur> listPredateur = new ArrayList<Predateur>();
    public static final double p_reproduce = 0.025;

    public Predateur(int posx, int posy, int energie, Terrain terrain) {
        super(posx, posy, energie, terrain);
        listPredateur.add(this);
    }

    public abstract double probaAttackSuccess(Poule p);
    public abstract void reproduce();

    public static ArrayList<Predateur> getListPredateur() { return listPredateur; }
}
```

```
public interface PredRat {
    // Uniquement là pour différencer le rat du renard avec un instanceof.
}
```

```
public interface PredRenard {
    // Uniquement là pour différencer le renard du rat avec un instanceof.
}
```

```
import java.util.ArrayList;

public class Rat extends Predateur implements PredRat{
    /*
     * C'est la classe des Rats, un prédateur.
     * Le renard se balade dans le terrain, en quête d'oeufs afin de les manger.
     * Il peut attaquer un oeuf proche de lui, mais se fera attaquer par les poulets autour de l'oeuf s'il y en a !
     * Il peut également se reproduire si le hasard le veut bien.
     */


    // Statistiques.
    private static int nbRatTotal = 0;
    private static int nbRatMorts = 0;
    private static int nbRatReproduction = 0;
    private static int nbOeufMange = 0;

    public Rat(int x, int y, Terrain terrain){
        super(x, y, 100, terrain);
        nbRatTotal++;
    }


    /*
     * Constructeur par copie, représentant la naissance d'un enfant renard.
     */
    public Rat(Rat parent){
        super(parent.x, parent.y, parent.energie/2 + 1, parent.terrain);
        nbRatTotal++;
    }



    public double probaAttackSuccess(Poule p){
        return (double)energie / ((double)p.energie + 1.0);
    }



    /*
     * Fonction correspondant à une attaque envers un oeuf.
     * Ici, s'il y a un oeuf sur la case du terrain correspondant à sa position, alors il y a tentative d'attaque.
     * S'il y a une poule autour de lui (distance euclidienne inférieure à 3), alors il y a une lutte.
     * S'il gagne la lutte, une autre poule l'attaque s'il y en a. S'il est toujours vivant, alors il mange l'oeuf et
rajoute à ses points d'énergie la quantité d'oeufs.
     * S'il perd la lutte, il meurt.
     */
    public void attack(ArrayList<Ressource> listRes){
        if (!terrain.caseEstVide(x, y)){
            Ressource res = terrain.getCase(x, y);
            if (res.type.equals("Oeufs")){
                for(Poule poule : Poule.getListPoules()){
```

```
                if (this.distance(poule.x, poule.y) < 6){
                    if (Math.random() < probaAttackSuccess(poule)){
                        listPredateur.remove(this);
                        listAgent.remove(this);
                        nbRatMorts++;
                        System.out.println("Echec de l'attaque du rat, il est mort !");
                        return;
                    }
                }
            }

            this.energie += res.getQuantite();
            nbOeufMange += res.getQuantite();

            listRes.remove(res);
            terrain.videCase(x, y);

            Ressource herbe = new Ressource("  __  ", 3);
            terrain.setCase(x, y, herbe);
            listRes.add(herbe);

            System.out.println("Un rat a mangé un ou plusieurs oeufs !");
        }
    }
}


/*
 * Fonction de reproduction du renard.
 * Si le hasard le veut bien, le renard fera un enfant qui a la même position que lui sur le terrain MAIS avec la moitié de son énergie.
 */
public void reproduce(){
    if (Math.random() < p_reproduce){
        Rat enfant = new Rat(this);
        listAgent.add(enfant);
        System.out.println("Un rat vient de naitre !");
    }
}



/*
 * Fonctions utilitaires pour les statistiques, le cas d'arrêt de la simulation...
 */
public static int getNbRatTotal() { return nbRatTotal; }
public static int getNbRatMorts() { return nbRatMorts; }
public static int getNbRatVivant() { return nbRatTotal - nbRatMorts; }
public static int getNbRatReproduction() { return nbRatReproduction; }

public boolean checkAlive(){
    if (energie <= 0){
```

```
            listPredateur.remove(this);
            listAgent.remove(this);
            nbRatMorts++;
            System.out.println("Un rat est mort d'épuisement !");
            return false;
        }

        return true;
    }



    /*
     * Fonction qui permet de récupérer les statistiques de fin de simulation des renards.
     */
    public static String stat(){
        String message = "Nombre de rats au total : " + nbRatTotal;
        message += "\nNombre de rats morts : " + nbRatMorts;
        message += "\nNombre d'oeufs mangés : " + nbOeufMange;
        return message;
    }
}
```

```
import java.util.ArrayList;
```

```
public class Renard extends Predateur implements PredRenard{
    /*
     * C'est la classe des Renards, un prédateur.
     * Le renard se balade dans le terrain, en quête de poules afin de les manger.
     * Il peut attaquer un poulet proche de lui, ainsi que manger un poulet mort d'épuissement (Ressource viande sur le terrain)
     * Il peut également se reproduire si le hasard le veut bien.
     */


    // Statistiques.
    private static int nbRenardTotal = 0;
    private static int nbRenardMorts = 0;
    private static int nbPouleMange = 0;
    private static int nbRenardReproduction = 0;

    public Renard(int x, int y, Terrain terrain){
        super(x, y, 100, terrain);
        nbRenardTotal++;
    }


    /*
     * Constructeur par copie, représentant la naissance d'un enfant renard.
     */
    public Renard(Renard parent){
        super(parent.x, parent.y, parent.energie/2 + 1, parent.terrain);
        nbRenardTotal++;
    }


    public double probaAttackSuccess(Poule p){
        return (double)energie / (double)p.energie;
    }



    /*
     * Fonction correspondant à une attaque envers une poule.
     * Ici, si la distance euclidienne entre lui et la poule est inférieure ou égale à 1, alors il y a tentative d'attaque.
     * Si la probabilité de succès lui est favorable, alors on rajoute les points d'énergie de la poule aux siens.
     * Il ne peut attaquer qu'une poule à la fois !
     */
    public void attack(){
        for(Poule poule : Poule.getListPoules()){
            if (this.distance(poule.x, poule.y) <= 1){
                if (Math.random() < probaAttackSuccess(poule)){
                    this.energie += poule.energie;
                    listAgent.remove(poule);
                    Poule.removeInListPoule(poule);
                    Poule.pouleTueCpt();
```

```
                nbPouleMange++;

                System.out.println("Un renard a mangé une poule !");
                return;
            }
        }
    }
}


/*
 * Fonction qui lui permet de manger la ressource Viande.
 * Si la case du terrain correspondant à sa position est munie d'une ou plusieurs viandes de poulet mort, alors il récupère 5 points d'énergie par viande.
 * L'herbe repousse sur le terrain avec 3 quantités.
 */
public void manger(ArrayList<Ressource> listRes){
    if (!terrain.caseEstVide(x, y)){
        Ressource res = terrain.getCase(x, y);
        if (res.type.equals("Viande")){
            this.energie += res.getQuantite()*5;
            listRes.remove(res);
            terrain.videCase(x, y);

            Ressource herbe = new Ressource("  __  ", 3);
            terrain.setCase(x, y, herbe);
            listRes.add(herbe);

            System.out.println("Un renard a mangé de la viande !");
        }
    }
}



/*
 * Fonction de reproduction du renard.
 * Si le hasard le veut bien, le renard fera un enfant qui a la même position que lui sur le terrain MAIS avec la moitié de son énergie.
 */
public void reproduce(){
    if (Math.random() < p_reproduce){
        Renard enfant = new Renard(this);
        listAgent.add(enfant);
        nbRenardReproduction++;
        System.out.println("Un renard vient de naitre !");
    }
}



/*
 * Fonctions utilitaires pour les statistiques, le cas d'arrêt de la simulation...
```

```
     */
    public static int getNbRenardTotal() { return nbRenardTotal; }
    public static int getNbRenardMorts() { return nbRenardMorts; }
    public static int getNbRenardVivant() { return nbRenardTotal - nbRenardMorts; }
    public static int getNbRenardReproduction() { return nbRenardReproduction; }
    public static int getNbPouleMange() { return nbPouleMange; }

    public boolean checkAlive(){
        if (energie <= 0){
            listPredateur.remove(this);
            listAgent.remove(this);
            nbRenardMorts++;
            System.out.println("Un renard est mort d'épuisement !");
            return false;
        }

        return true;
    }



    /*
     * Fonction qui permet de récupérer les statistiques de fin de simulation des renards.
     */
    public static String stat(){
        String message = "Nombre de renards au total : " + nbRenardTotal;
        message += "\nNombre de renards morts : " + nbRenardMorts;
        message += "\nNombre de poulet mangés : " + nbPouleMange;
        return message;
    }
}
```

```
import java.util.ArrayList;
```

```
public class Simulation {
    // Initialisation des variables globales
    public static final String couleur = "\u001B[34m";
    public static final String couleurReset = "\u001B[0m";
    public static final String separateur = couleur +
"=========================================================================" +
couleurReset;
    public static final double p_eclosion = 0.10;

    private Terrain terrain;
    private ArrayList<Agent> listAgent;
    private ArrayList<Predateur> listPredateur;
    private ArrayList<Ressource> listRes;


    // Construction de la simulation
    public Simulation(int m, int n){
        terrain = new Terrain(10,20);
        listAgent = Agent.getListAgent();
        listPredateur = Predateur.getListPredateur();
        listRes = new ArrayList<Ressource>();

        // On rajoute, au hasard, m oeufs sur le terrain.
        for(int k = 0; k < m; k++){
            int x = (int)(Math.random() * terrain.nbLignes);
            int y = (int)(Math.random() * terrain.nbColonnes);

            if (terrain.caseEstVide(x, y)){
                Ressource oeuf = new Ressource("Oeufs", 1);
                listRes.add(oeuf);
                terrain.setCase(x, y, oeuf);
            }
            else{
                Ressource oeuf = terrain.getCase(x, y);
                oeuf.setQuantite(oeuf.getQuantite() + 1);
            }
        }


        // Sur le reste du terrain, on lui rajoute une parcelle d'herbe, initialisé à 5 quantités.
        for(int i = 0; i < terrain.nbLignes; i++){
            for(int j = 0; j < terrain.nbColonnes; j++){
                if (terrain.caseEstVide(i,j)){
                    Ressource herbe = new Ressource("  __  ", 5);
                    listRes.add(herbe);
                    terrain.setCase(i, j, herbe);
                }
            }
        }
```

```
        // On crée 5*n poules, 2*n renards et 4*n rats avec des positions aléatoires.
        for(int k = 0; k < n; k++){
            int x = (int)(Math.random() * terrain.nbLignes);
            int y = (int)(Math.random() * terrain.nbColonnes);

            for(int cpt = 0; cpt < 5; cpt++)
                new Poule(x, y, terrain);

            x = (int)(Math.random() * terrain.nbLignes);
            y = (int)(Math.random() * terrain.nbColonnes);

            for(int cpt = 0; cpt < 2; cpt++)
                new Renard(x, y, terrain);

            x = (int)(Math.random() * terrain.nbLignes);
            y = (int)(Math.random() * terrain.nbColonnes);

            for(int cpt = 0; cpt < 4; cpt++)
                new Rat(x, y, terrain);
        }
    }


    // Cette fonction permet de passer à la prochaine itération de la simulation.
    public void rafraichir(){

        // On efface le terminal.
        System.out.print("\033\143");

        System.out.println("\u001B[32m" + "Simulation de la vie des poules et des prédateurs (ici les renards et
les rats)." + Simulation.couleurReset);
        System.out.println("\n"+ separateur +"\n");

        // On modifie aléatoirement la direction des agents, et on les fait se déplacer.
        listAgent = new ArrayList<Agent>(Agent.getListAgent());
        for(Agent a : listAgent){
            a.changeDirection();
            int newx = a.getX() + a.getDx();
            int newy = a.getY() + a.getDy();
            a.seDeplacer(newx, newy);
        }




        /*
         * Premièrement, on regarde si la poule est morte.
         *
         * Si elle ne l'est pas, elle mange l'herbe au sol s'il y en a, puis elle pondra un oeuf si le hasard l'accepte.
         * Sinon, un bout de viande est déposé sur le terrain à sa position. Elle permettra au renard de se nourrir.
         *
         */
```

```
        ArrayList<Poule> listPoule = new ArrayList<Poule>(Poule.getListPoules());
        for(Poule poule : listPoule){
            if (poule.checkAlive()){
                poule.manger();
                poule.pondre(listRes);
            }

            else {
                Ressource viande = new Ressource("Viande", 1);
                listRes.add(viande);

                // Si la poule est sur un oeuf, au moment de sa chute, l'oeuf se détruit.
                if (terrain.caseEstVide(poule.getX(), poule.getY()))
                    terrain.setCase(poule.getX(), poule.getY(), viande);
                else{
                    Ressource actu = terrain.getCase(poule.getX(), poule.getY());
                    listRes.remove(actu);
                    terrain.videCase(poule.getX(), poule.getY());
                    terrain.setCase(poule.getX(), poule.getY(), viande);
                }
            }
        }

        System.out.println("\n" + separateur + "\n");




        /*
         * On regarde si les prédateurs (Renards et Rats) sont morts.
         *
         * Si oui, les renards mangent la viande à leur pied s'il y en a, et attaque une poule à proximité s'il y en a
une.
         * Le rat attaque un oeuf à proximité s'il y en a un.
         * Enfin, il y a reproduction des prédateurs si le hasard le veut bien.
         *
         */
        listPredateur = new ArrayList<Predateur>(Predateur.getListPredateur());
        for(Predateur predateur : listPredateur){
            if (predateur.checkAlive()){
                if (predateur instanceof PredRenard){
                    ((Renard)predateur).manger(listRes);
                    ((Renard)predateur).attack();
                }

                else
                    ((Rat)predateur).attack(listRes);

                predateur.reproduce();
            }
        }
```

```
        System.out.println("\n"+ separateur + "\n");



        /*
         * NOTE : "  __  " CORRESPOND A L'HERBE.
         *
         * Ici, nous parcourons chaque ressource du terrain.
         * La quantité d'herbe augmente de 1.
         * L'oeuf a une probabilité qu'elle puisse éclore. Si elle éclot, une poule est créee et la quantité d'oeuf baisse de 1 (ou disparait s'il était seul).
         * L'herbe revient sur la case de l'oeuf après éclosion.
         *
         */
        ArrayList<Ressource> temp = new ArrayList<Ressource>(listRes);
        for(Ressource res : temp){
            if (res.type.equals("  __  "))
                res.setQuantite(res.getQuantite() + 1);

            else if (res.type.equals("Oeufs") && (Math.random() < p_eclosion)){
                Poule poule = new Poule(res.getX(), res.getY(), terrain);

                if (res.getQuantite() == 1){
                    int x_herbe = res.getX(), y_herbe = res.getY();
                    terrain.videCase(res.getX(), res.getY());

                    Ressource herbe = new Ressource("  __  ", 3);
                    terrain.setCase(x_herbe, y_herbe, herbe);
                    listRes.remove(res);
                    listRes.add(herbe);
                }

                else { res.setQuantite(res.getQuantite() - 1); }

                Poule.oeufEclos();
                System.out.println("Un oeuf vient d'éclore en (" + poule.getX() + "," + poule.getY() +") !");
            }
        }

        System.out.println("\n"+ separateur + "\n");


        // On affiche le terrain après toutes les modifications apportées.
        terrain.affiche(6);
    }



    /*
     * Cette fonction affiche les statistiques à la fin de la simulation.
     */
```

```
    public static String stat(){
        String message = separateur + "\n\n";
        message += Poule.stat();
        message += "\n\n" +separateur + "\n\n";
        message += Renard.stat();
        message += "\n\n" + separateur + "\n\n";
        message += Rat.stat();
        message += "\n\n" + separateur + "\n";
        message += "PINHO FERNANDES Enzo - 21107465\n";
        message += "DURBIN Deniz Ali - 21107465";

        return message;
    }
}
```

```
import java.lang.Thread;

public class TestSimulation {
   public static void main(String[] args){
      Simulation simulation = new Simulation(10, 5);


      int nbIteration = 0;
      while(nbIteration < 60){
         simulation.rafraichir();
         System.out.println("Nombre de poules : " + Poule.getNbPouleVivant());
         System.out.println("Nombre de renards : " + Renard.getNbRenardVivant());
         System.out.println("Nombre de rats : " + Rat.getNbRatVivant());
         System.out.println("\n" + Simulation.separateur + "\n");

         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
         }

         nbIteration++;
      }

      System.out.print("\033\143");
      System.out.println(Simulation.stat());
      System.out.println("\nFin de simulation !");
   }
}
```