Mini-projet LU2IN002 - 2020-2021

Nom : GIULIANI	*Nom :*
Prénom : Carla	*Prénom :*
N° étudiant : 28605022	*N° étudiant :*

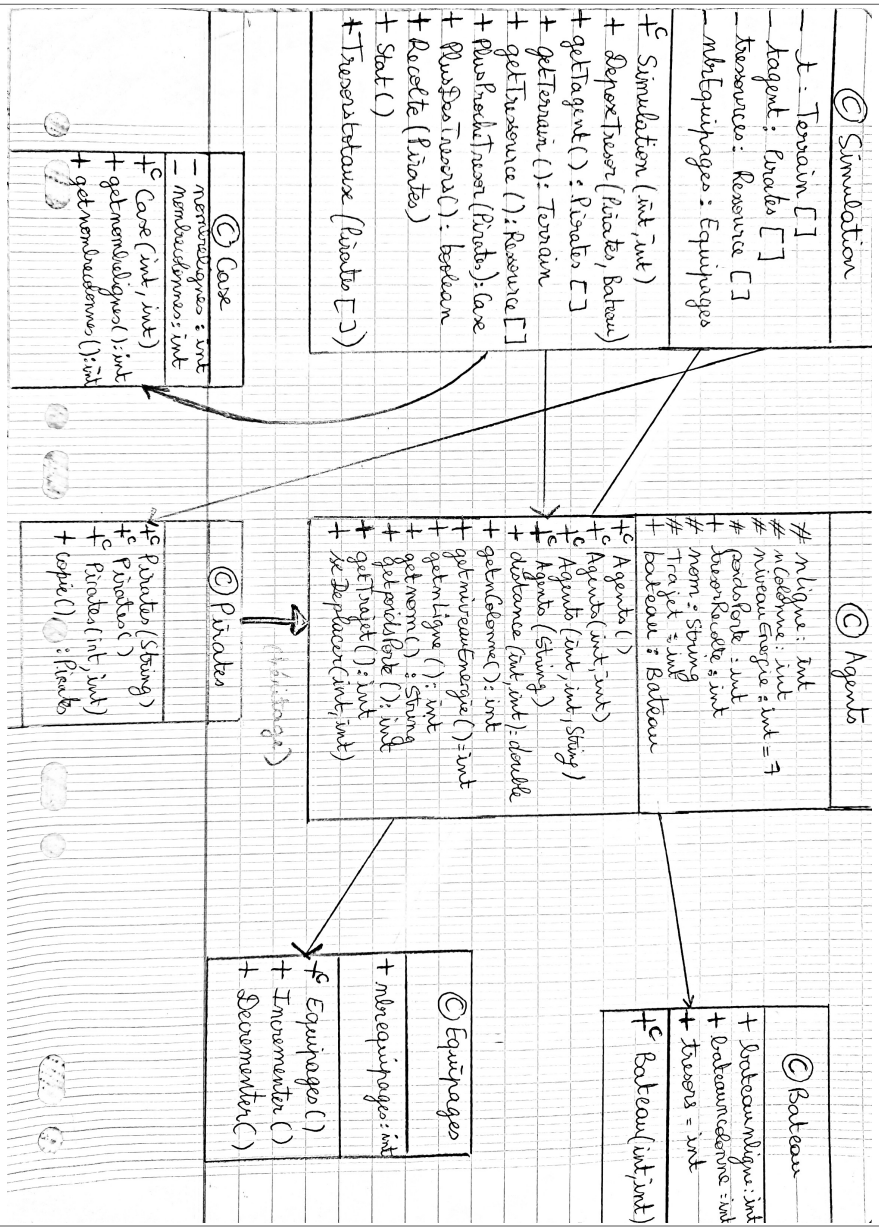
Thème de simulation choisi (en 2 lignes max.)
Plusieurs équipages de pirates prennent escale sur une île mystérieuse où ils tenteront de trouver des fabuleux trésors afin de s'enrichir.

Description des classes et de leur rôle dans la simulation (2 lignes max par classe)
Classe Agents: la classe fait référence aux individus afin de les déplacer sur un terrain et également de pouvoir calculer une distance. Elle permet la création et le déplacement des pirates. Classe Pirates: Cette classe héritée de la classe Agents créer des agents qui sont des pirates. Elle permet de générer des pirates. Classe Terrain: permet la création de notre monde (ici l'île) et de mettre à jour les données présentes dans ses cases. Classe Ressource: permet de créer et mettre à jour les ressources sur le terrain (la position des ressources, la quantité de celles-ci). Classe Simulation: Elle fait une simulation de pirates récoltants des trésors. Elle permet de savoir où se trouve le prochain trésor, de le récolter et de le déposer dans son bateau et affiche les statistiques. Classe Equipages: Elle permet de nous donner le nombre d'équipages présent sur l'île. Classe Bateau: Elle permet de savoir où se situe le bateau et combien de trésors y sont rangés. Classe Case: Elle permet de donner la case où le pirate doit se rendre. Classe TestSimulation: Elle permet de simuler la récolte des trésors par les pirates sur l'île.

Décrire, dans les grandes lignes, ce qui se passe durant la simulation (max. 5-6 lignes)

Durant la simulation, des pirates arrivent sur les bords de l'île puis grâce à leur carte magique, ils ont accès aux coordonnées du trésor le plus proche. Ils décident donc de se déplacer vers ces coordonnées pour récupérer le trésor. Sachant qu'ils n'ont le droit qu'à 7 trajets chacun. Si le trajet est long, ils feront des pauses afin de récupérer de l'énergie. Une fois un trésor trouvé il le ramèneront au bateau (les coordonnées du bateau seront celle du pirate à l'arrivée) et s'ils n'ont pas dépassés le nombre de trajet, ils pourront repartir à la recherche d'un nouveau trésor.

Schéma UML fournisseur des classes (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 liste d'objets	Classe Simulation

Classe statique contenant que des méthodes statiques	Equipages
Héritage	Classe Pirates (hérite de la classe Agents)
Classe avec composition	Simulation, TestSimulation
Classe avec un constructeur par copie ou clone()	Pirates
Noms des classes créées (entre 4 et 10 classes)	Classe Agents Classe Simulation Classe Pirates Classe Case Classe TestSimulation Classe Equipages Classe Bateau

Copier / coller de vos classes à partir d'ici :

CLASSE AGENTS :

```
import java.util.Random;
/**
 * Classe Agents, fait référence aux individus  afin de les
déplacer sur un terrain et également de pouvoir calculer une
distance
 */

/**
 * @author Carla Giuliani (Mini-projet du 16/11/2020)
 *
 */

 public class Agents{

    protected int nLigne, nColonne;
    protected int niveauEnergie=7;
    protected int poidsPorte;
    public int tresorRecolte;
    protected String nom;
    protected int Trajet;
    public Bateau bateau;
    /**
     * initialise aléatoirement les coordonnées de l'objet courant
au bord de l'île, l'énergie à 7 points, le nombre de trésors
récoltés à 0 kg et le poids porté aléatoirement entre 0 et 30 kg
     */
    public Agents(){
        nLigne=(int)(Math.random()*11);
        if(nLigne!=0){
            nColonne=0;
        }
        else{
            nColonne=(int)(Math.random()*11);
        }
        tresorRecolte=0;
```

```
        poidsPorte=(int)(Math.random()*31);

        Random rand = new Random();
        String str="";
       for(int i = 0 ; i < 5 ; i++){
            char c = (char)(rand.nextInt(26) + 97);
            str += c;

        }
        nom=str;
        bateau = new Bateau(nLigne, nColonne);
        Trajet=0;
    }
    /**
     *
     * @param nom
     * nom du pirate
     */
    public Agents(String nom){
        nLigne=(int)(Math.random()*11);
        if(nLigne!=0){
            nColonne=0;
        }
        else{
            nColonne=(int)(Math.random()*11);
        }
        tresorRecolte=0;
        poidsPorte=(int)(Math.random()*31);
        this.nom=nom;
        bateau = new Bateau(nLigne, nColonne);
        Trajet=0;
    }
    /**
     *
     * @param nligne
     * @param ncolonne
     * coordonnées du pirate
     */
    public Agents(int nligne, int ncolonne){
        nLigne=nligne;
```

```
        nColonne=ncolonne;
        tresorRecolte=0;
        poidsPorte=(int)(Math.random()*31);

        Random rand = new Random();
        String str="";
       for(int i = 0 ; i < 5 ; i++){
            char c = (char)(rand.nextInt(26) + 97);
            str += c;

        }
        nom=str;
        bateau = new Bateau(nLigne, nColonne);
        Trajet=0;

    }
    /**
     *
     * @param nligne
     * @param ncolonne
     * coordonnées du pirates
     * @param nom
     * nom du pirate
     */
    public Agents(int nligne, int ncolonne,String nom){
        nLigne=nligne;
        nColonne=ncolonne;
        tresorRecolte=0;
        poidsPorte=(int)(Math.random()*31);
        this.nom=nom;
        bateau = new Bateau(nLigne, nColonne);
        Trajet=0;
    }
    /**
     * cette méthode sert à déplacer les agents sur le terrain
     * @param xnew
     * nouvelle coordonnée nLigne de l'objet courant
     * @param ynew
     * nouvelle coordonnée nColonne de l'objet courant

```

```
     */
    public void seDeplacer(int xnew,int ynew){
        System.out.println("Le pirate va se déplacer au
coordonnées ("+ xnew+","+ynew+")");
        System.out.println("\n");

        if(xnew==nLigne && ynew==nColonne){
            System.out.println("Le pirate est déjà au coordonnées
indiquées");
        }

        if(Trajet<7){
            if(xnew>nLigne && ynew==nColonne){
                while(xnew>nLigne && niveauEnergie>0){
                    nLigne++;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }

            }
            if(xnew==nLigne && ynew>nColonne){
                while(ynew>nColonne && niveauEnergie>0){
                    nColonne++;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }
```

```
            }
            if(xnew>nLigne && ynew>nColonne){
                if(xnew==ynew && nLigne==nColonne){
                    while(ynew>nColonne && niveauEnergie>0){
                        nColonne++;
                        nLigne++;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }
                    }
                }
                else{
                    while(ynew>nColonne && niveauEnergie>0){
                        nColonne++;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }

                    }
                    while(xnew>nLigne && niveauEnergie>0){
                        nLigne++;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }

                    }
                }
```

```
            }

            if(xnew<nLigne && ynew==nColonne){
                while(xnew<nLigne && niveauEnergie>0){
                    nLigne--;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }

            }
            if(xnew==nLigne && ynew<nColonne){
                while(ynew<nColonne && niveauEnergie>0){
                    nColonne--;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }
            }
            if(xnew<nLigne && ynew>nColonne){
                while(ynew>nColonne && niveauEnergie>0){
                    nColonne++;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

```

```
                }
                while(xnew<nLigne && niveauEnergie>0){
                    nLigne--;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }
            }
            if(xnew>nLigne && ynew<nColonne){
                while(ynew<nColonne && niveauEnergie>0){
                    nColonne--;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }
                while(xnew>nLigne && niveauEnergie>0){
                    nLigne++;
                    niveauEnergie--;
                    if(tresorRecolte>(poidsPorte/4)){
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte*(2/4))){
                            niveauEnergie--;
                        }
                    }

                }
            }

            if(xnew<nLigne && ynew<nColonne){
                if(xnew==ynew && nLigne==nColonne){
```

```
                    while(ynew<nColonne && niveauEnergie>0){
                        nColonne--;
                        nLigne--;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }

                    }
                }
                else{
                    while(ynew<nColonne && niveauEnergie>0){
                        nColonne--;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }

                    }
                    while(xnew<nLigne && niveauEnergie>0){
                        nLigne--;
                        niveauEnergie--;
                        if(tresorRecolte>(poidsPorte/4)){
                            niveauEnergie--;
                            if(tresorRecolte>(poidsPorte*(2/4))){
                                niveauEnergie--;
                            }
                        }
                    }
                }
            }
            Trajet++;
            if(niveauEnergie<=0){
```

```
                System.out.println("Le pirate est épuisé, il fait
une pause à la case ("+nLigne+','+nColonne+')');
                niveauEnergie=7;

            }
        }


    }
    /**
     * Accesseur du niveauEnergie
     * @return niveauEnergie
     *
    */
    public int getniveauEnergie(){
        return niveauEnergie;
    }
    /**
     * Accesseur de nLigne
     * @return nLigne
     * */
    public int getnLigne(){
        return nLigne;
    }
    /**
     * Accesseur de nColonne
     * @return nColonne*/
    public int getnColonne(){
        return nColonne;
    }
    /**
     * Accesseur de poidsPorte
     * @return poidsPorte
     * */
    public int getpoidsPorte(){
        return poidsPorte;
    }

    /**
     * Accesseur du nom
```

```
     * @return nom
     *
    */
    public String getnom(){
        return nom;
    }
      /**
       * Accesseur du Trajet
       * @return Trajet
       *
       * */
      public int getTrajet(){
        return Trajet;
    }
    /**
     * @param x
     * coordonnée nLigne d'un objet
     * @param y
     * coordonnée nColonne d'un objet
     * @return Cette méthode renvoit la distance euclienne entre
l'objet courant et la case de coordonnées passées en paramètre
     */
    public double distance(int x,int y){
        return Math.sqrt((x-nLigne)*(x-nLigne)+(y-nColonne)*(y-
nColonne));
    }

}
```

CLASSE PIRATES :

```java
/**
 *Classe Pirates hérité de la classe Agents, créer des agents qui sont des pirates
 */

/**
 * @author Carla Giuliani (Mini-projet du 16/11/2020)
 *
 */
public class Pirates extends Agents{
    /**
     * initialise le pirate
     *@param nom
     * nom du pirate
     */
    public Pirates(String nom){
        super(nom);
    }

    /**
     * initialise le pirate
     */
    public Pirates(){
        super();
    }
    /**
     * initialise le pirate
     * @param ligne
     * @param colonne
     * coordonnées du pirates
     */
    public Pirates(int ligne, int colonne){
        super(ligne,colonne);
    }

    /**
```

```
     * méthode qui copie le pirate (objet courant) pour créer un
nouveau pirate aux mêmes coordonnées
     * @return un pirate
     */
    public Pirates copie(){
        return new Pirates(this.nLigne,this.nColonne);
    }
}
```

CLASSE CASE :

```java
/**
 * @author Carla Giuliani (Mini-projet du 16/11/2020)
 *
 */
public class Case{

    private int nombrelignes,nombrecolonnes;

    /**
     * initialise la case avec les valeurs x et y
     * @param x
     * @param y
     */
    public Case(int x,int y){
        this.nombrelignes=x;
        this.nombrecolonnes=y;
    }

    /**
     * Accesseur du nombre de lignes
     * @return nombrelignes
     *
     */
    public int getnombrelignes(){
        return nombrelignes;
    }

    /**
     * Accesseur du nombre de colonnes
     * @return nombrecolonnes
     */
    public int getnombrecolonnes(){
        return nombrecolonnes;
    }
}
```

CLASSE EQUIPAGES :

```
/**
 *Classe Equipage, calcule le nombre d'équipages
 */

/**
 * @author Carla Giuliani (Mini-projet du 16/11/2020)
 *
 */
public class Equipages {

    public static int nbrequipages;

    /**
     * initialise le nombre d'équipages à 0
     */
    public Equipages(){
        nbrequipages=0;
    }

    /**
     * permet d'incrémenter de 1 le nombre d'équipage
     */
    public static void Incrementer(){
        nbrequipages++;
    }

    /**
     * permet de décrémenter de 1 le nombre d'équipages
     */
    public static void Decrementer(){
        nbrequipages--;
    }


}
```

CLASSE BATEAU :

```java
public class Bateau {
    /**
     * @param bateaunLigne
     * @param bateaunColonne
     * coordonnées du bateau du pirate
     * @param tresors
     * quantité de trésors sur le bateau
     */
    public int bateaunLigne,bateaunColonne;
    public int tresors;

    /**
     * initialise le  caractéristiques du bateau
     * @param x
     * correspond à bateauunLigne
     * @param y
     * correspond à bateauunColonne
     */
    public Bateau(int x , int y){
        bateaunLigne=x;
        bateaunColonne=y;
        tresors=0;
    }

}
```

CLASSE SIMULATION:

```java
/**
 *Classe Simulation, fait une simulation de pirates récoltants des trésors.
 */

/**
 * @author Carla Giuliani (Mini-projet du 16/11/2020)
 *
 */

public class Simulation{
    private Terrain  t= new Terrain(10,10);
    private Pirates [] tagent;
    private Ressource [] tressource;
    private Equipages nbrEquipages=new Equipages();

    /**
     * initialise le tableau de ressources avec un poids aléatoire (qui équivaut à la quantité) puis place ces resources aléatoirement sur le terrain (ici l'île) et initalise le tableau d'agents.
     * @param m
     * nombre de ressources, taille du tableau de ressources
     * @param n
     * nombre d'agents, taille du tableau d'agents
     */
   public Simulation(int m, int n){
       tressource= new Ressource[m];
       for(int i=0; i<tressource.length ;i++){
            int quantite=(int)(Math.random()*21);
            if(quantite!=0){
                tressource[i]= new Ressource("trésor",quantite);
            }
            else{
                while(quantite==0){
                    quantite=(int)(Math.random()*21);
                }
                tressource[i]= new Ressource("trésor",quantite);
```

```
            }

        }
        for(int j=0;j<tressource.length;j++){
             int x=(int)(Math.random()*9);
             int y=(int)(Math.random()*9);
         if(t.caseEstVide(x,y)==true){
             t.setCase(x,y,tressource[j]);
         }
         else{
             while(t.caseEstVide(x,y)!=true){
                 x=(int)(Math.random()*9);
                 y=(int)(Math.random()*9);
             }
             t.setCase(x,y,tressource[j]);
         }

        }
        tagent= new Pirates[n];
        for(int k=0; k<tagent.length;k++){
            if((int)(Math.random()%11)==4 && k>0){

                tagent[k]= tagent[k-1].copie();

            }
            else{
             tagent[k]= new Pirates();

            }
        }

    }

    /**
     * Accesseur du terrain
     * @return t
     */
```

```java
    public Terrain getTerrain(){
        return t;
    }
    /**
     * Accesseur du tableau d'agents
     * @return tagent
     */
    public Pirates [] getTagent(){
        return tagent;
    }
    /**
     * Accesseur du tableau de ressources
     * @return tressource
     */
    public Ressource [] getTressource(){
     return tressource;
}

    /**
     *
     * @param pirate
     * agent avec toutes ses caractéristiques
     * @return les coordonnées du trésor le plus proche
     */
    public Case PlusProcheTresor(Pirates pirate){
        double dist=-1;
        int nblig=0;
        int nbcol=0;
        for(int h=0;h<t.nbColonnes;h++){
            for(int p=0;p<t.nbLignes;p++){
                if((t.caseEstVide(p,h))==false){
                    if(pirate.distance(p,h)<dist||dist==-1){
                        dist=pirate.distance(p,h);
                        nblig=p;
                        nbcol=h;
                    }
                }
            }

        }
```

```
        System.out.println("le trésor le plus proche se trouve à la
case ("+nblig+","+nbcol+").");
        Case c= new Case(nblig,nbcol);
        return c;

    }
    /**
     * @return renvoit true s'il n'y a plus de trésors sur le
terrain sinon false
     */
    public boolean PlusDesTresors(){
        for(int i=0;i<t.nbLignes;i++){
             for(int j=0;j<t.nbColonnes;j++){
                 if(t.caseEstVide(i,j)==false){
                     return false;
                 }
             }
         }
         return true;
    }

    /**
     * Cette méthode permet de récupérer les ressources sur soi et
ainsi les retirer du terrain
     * @param agent
     * agent correspondant à un pirate avec toutes ses
caractéristiques
     */
    public void Recolte(Pirates agent){

if(t.caseEstVide(agent.getnLigne(),agent.getnColonne())==false){

if((t.getCase(agent.getnLigne(),agent.getnColonne()).getQuantite()
>agent.getpoidsPorte())){


t.getCase(agent.getnLigne(),agent.getnColonne()).setQuantite((t.ge
tCase(agent.getnLigne(),agent.getnColonne()).getQuantite())-
((agent.getpoidsPorte())));
```

```
                agent.tresorRecolte= (agent.getpoidsPorte());

            }

if((t.getCase(agent.getnLigne(),agent.getnColonne()).getQuantite()
+agent.tresorRecolte)<=agent.getpoidsPorte()){


agent.tresorRecolte=(t.getCase(agent.getnLigne(),agent.getnColonne
()).getQuantite());

                Ressource
r=(t.videCase(agent.getnLigne(),agent.getnColonne()));
            }
        }
    }

    /**
     * méthode qui permet de déposer les trésors que l'on a sur soi
     * @param pirate
     * @param bateau
     */
    public void DeposeTresor(Pirates pirate, Bateau bateau){
        pirate.bateau.tresors+= pirate.tresorRecolte;

        pirate.tresorRecolte=0;

    }

    /**
     * affiche le pourcentage et le nombre de ressources récoltées
     */
    public void Stat(){
        int rest=0;
        int nbrressources=tressource.length;

        for(int o=0;o<t.nbColonnes;o++){
         for(int q=0;q<t.nbLignes;q++){
```

```
            if((t.caseEstVide(q,o))==false){
                rest++;
            }
        }
        }
        int recolte= nbrressources-rest;
        double pourcentage = (recolte*1.0/nbrressources*1.0)*100;
        System.out.println(pourcentage+ " % des ressources ont été
récolté totalement (plus aucun trésor à cette case) soit "+recolte
+" sur "+nbrressources+" ressources sur le terrain");
    }
    /**
     * Cette méthode affiche la quantité de trésor amassé sur son
bateau par chaque par chaque pirate
     * @param tagent
     */
    public void Tresorstotaux(Pirates [] tagent){
        int [] tab=new int[tagent.length];
        int k=0;
        String str="";
        boolean bool=false;
        boolean passe=false;
        for(int i=0;i<tagent.length;i++){
            str=tagent[i].getnom();
            for(int j=i+1;j<tagent.length;j++){
                if(tagent[i].bateau.bateaunLigne==
tagent[j].bateau.bateaunLigne && tagent[i].bateau.bateaunColonne==
tagent[j].bateau.bateaunColonne){


tagent[i].bateau.tresors+=tagent[j].bateau.tresors;

                    str+= ", "+ tagent[j].getnom();
                    tab[k]=j;
                    k++;
                    passe=true;

                }
            }
            for(int l=0;l<k;l++){
```

```
                if(tab[l]==i){
                    bool=true;
                }
            }
            if(bool==false){
                nbrEquipages.Incrementer();
                if(passe==true){
                    System.out.println("les pirates "+ str +" ont
récolté au total " + tagent[i].bateau.tresors);
                }
                else{
                    System.out.println("le pirate "+ str +" a
récolté au total " + tagent[i].bateau.tresors );
                }
            }
            str="";
            bool=false;
            passe=false;
        }
        System.out.println("\n");
        System.out.println("il y a "+nbrEquipages.nbrequipages +"
d'équipages sur l'île");

        }
}
```

CLASSE TESTSIMULATION:

```
public class TestSimulation{
    /**
     * @param args
     */
    public static void main(String[] args) {

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

        // création de notre simulation

        Simulation s= new Simulation(20,15);

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

        //affichage du terrain avant l'arrivée des pirates

        s.getTerrain().affiche();

        Pirates [] agents=s.getTagent();
        Ressource [] ressources=s.getTressource();

        // affichage des ressources et de leur position
        for(int j=0;j<ressources.length;j++){
            System.out.println(ressources[j].toString());

        }
        System.out.println("\n");
        System.out.println("\n");

        //affichage des noms des pirates avec le poids maximum
pouvant être porté
        for(int k=0;k<agents.length;k++){
            System.out.println("le pirate "+agents[k].getnom()+ "
peut porter maximum "+agents[k].getpoidsPorte()+ " kilos de
trésors.");

        }
```

```
        //simulaion des déplacement avec récoltes de trésors et
retour au bateau

        int cpt=0;

        while(cpt<agents.length){
            cpt=0;

            for(int i=0; i<agents.length;i++){

                if((agents[i].getTrajet())<7){

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

                    System.out.println("C'est moi le pirate :
"+agents[i].getnom());

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

                    System.out.println("avant son trajet le pirate
est aux coordonnées ("+ agents[i].getnLigne()+ ", "+
agents[i].getnColonne()+")");

                    //si le pirate a trouvé un trésor, il
s'empresse de le ramener à son bateau

                    if(agents[i].tresorRecolte!=0){
                        System.out.println("le pirate retourne au
bateau");

agents[i].seDeplacer(agents[i].bateau.bateaunLigne,agents[i].batea
u.bateaunColonne);
                    }

                    // sinon il continue ses recherches
                    else{
                        Case c=s.PlusProcheTresor(agents[i]);
```

```
agents[i].seDeplacer(c.getnombrelignes(),c.getnombrecolonnes());
                        s.Recolte(agents[i]);
                    }

                    System.out.println("le pirate après son trajet
est aux coordonnées ("+agents[i].getnLigne()
+","+agents[i].getnColonne()+") et il lui reste "+
agents[i].getniveauEnergie()+ " points d'énergie");

                    // s'il est arrivé à son bateau il dépose tous
les trésors

                    if(agents[i].tresorRecolte!=0 &&
(agents[i].getnLigne()==agents[i].bateau.bateaunLigne &&
agents[i].getnColonne() ==agents[i].bateau.bateaunColonne)){

                        s.DeposeTresor(agents[i],
agents[i].bateau);
                    }

                    System.out.println("nbr de trésors sur le
pirate : "+ agents[i].tresorRecolte );
                    System.out.println("nbr de trésors dans son
bateau: "+ agents[i].bateau.tresors);
                }
                else{
                    cpt++;
                }
            }
        }

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

        // affichage des satistiques
        s.Stat();

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

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

        // affichage des trésors récoltés par les pirates
        s.Tresorstotaux(agents);

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

        // affichage des trajets finaux des pirates

        for(int h=0;h<agents.length;h++){
            System.out.println("nbr de trajets finaux:
"+agents[h].getTrajet());
        }

        // affichage du terrain après le passage des pirates
        s.getTerrain().affiche();
    }
}
```