Mini-projet LU2IN002 - 2020-2021

Nom : GRONDIN	*Nom :* Bouaicha
Prénom : Laurent	*Prénom : Kenza*
N° étudiant : 2767308	*N° étudiant :* 3703308

Thème de simulation choisi (en 2 lignes max.)

Simulation de la récolte de pépites d'or par des mineurs.
Les pépites d'or se trouvent en quantité aléatoire dans des ressources (minerais) disposées aléatoirement sur un terrain.

Description des classes et de leur rôle dans la simulation (2 lignes max par classe)

- La classe Bao (Boite A Outils) est la classe statique fournissant des méthodes statiques permettant d'assurer diverses fonctionnalités (tirage d'un nombre àléatoire, verifier qu'un nombre est dans un intervalle…) .
- La classe Mineur permet de définir les agents ainsi que leurs actions de déplacement sur le terrain.
- La classe Ressource permet de définir les ressources devant être détruites afin de collecter l'or.
- La classe Terrain permet de delimiter l'environnement de la simulation, ainsi que de representer les objets Ressources et Hérités.
- La classes Gravats est une classe fille de ressource permettant de representer une ressource entierement détruite sur le terrain (et donc ne contenant plus de pepite d'or)
- La classe Simulation permet de lancer la simulation ainsi que d'établir les régles la régissant:(
    - Phase d'initialisation :
        - Génére et place aléatoirement des quantités de Ressources sur le terrain
        - Génére par copie des agents (Mineur)
    - Phase de Recherche & Recolte
        - Définie les actions et interactions possibles entre agents et ressource à chaque tour.
- La classe TestSimulation permet de lancer les differents tests en permettant ainsi à l'utilisateur de choisir les parametres qu'il souhaite appliquer.C'est elle qui permet la dernère phade de la simulation :
    - Phase Bilan :
        - Affichage des statistiques de récoltes par mineur ainsi que du terrain en fin de simulation.

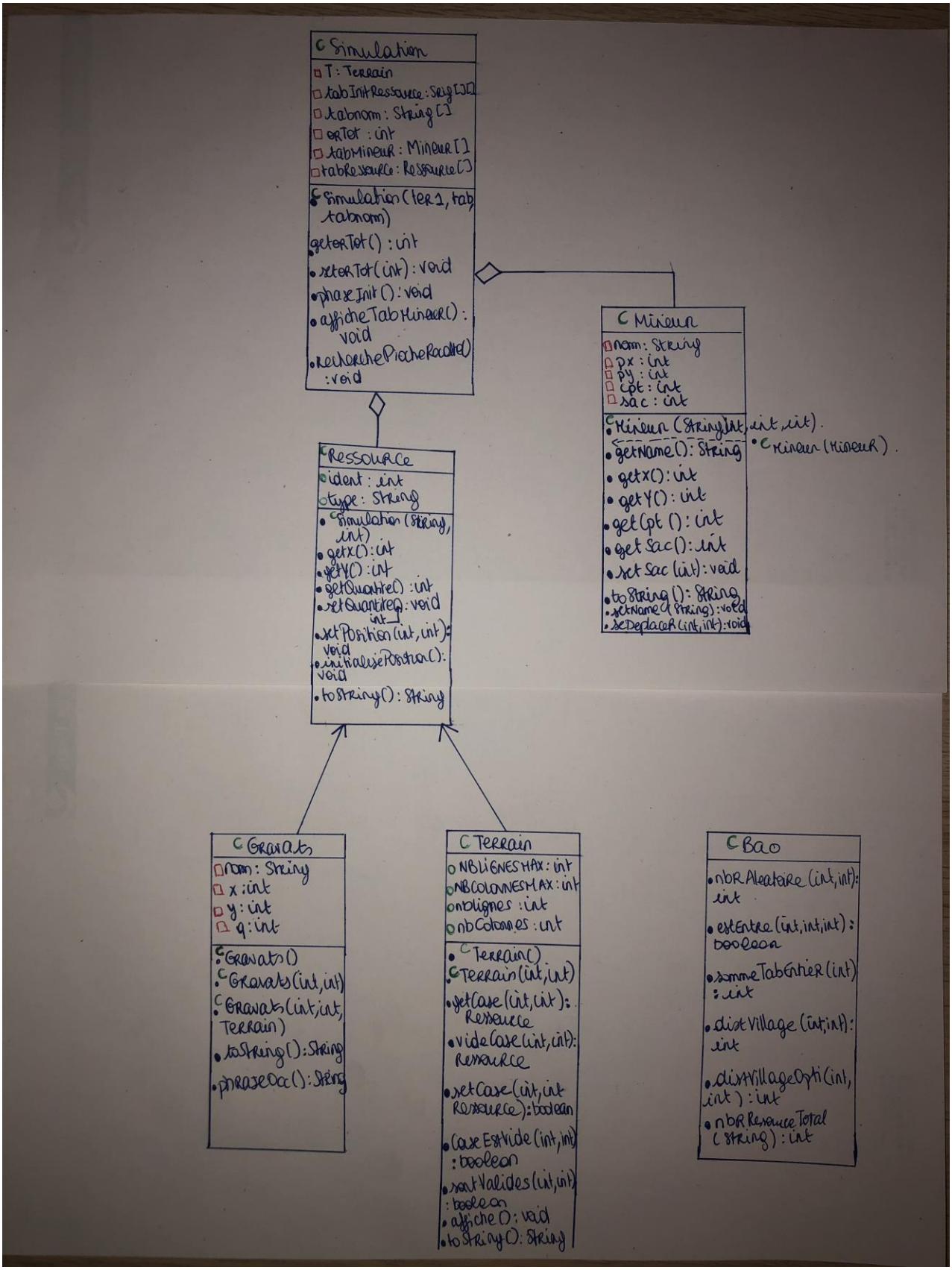
Décrire, dans les grandes lignes, ce qui se passe durant la simulation (max. 5-6 lignes)

- Phase d'initialisation : Le terrain est généré, puis les ressources y sont places aléatoirements.Les mineurs sont ensuite créés par copie (à partir de l'objet chef (du village)

qui lui ne participera pas à la récolte.)

- Phase de Recherche&Recolte : Cette phase commence par l'introduction des mineurs sur le terrain. Il feront chacun un coup par tour . Un coup peut être soit un déplacement, soit une récolte .Chaque pépite collectée est place dans le sac du mineur.Lorsqu'une ressource est entièrement vidée de ses pépites, elle est considérée comme détruite, et est alors remplacée par l'objet Gravas sur le terrain. Les déplacements des mineurs sur le terrain sont aléatoire et de portée d'une case (pas de sur place ni de saut de case possible).Cette phase se stop alors lorsque toutes les pépites disponibles sont récoltées.
- Phase Bilan: Est alors produit un bilan de la simulation indiquant le nombre de tour ayant été necessaire pour récolter l'ensemble des pépites ainsi que la repartition de cette récolte entre les agents.(nombre de pépites colléctées par chaque mineur). Le terrain est lui aussi afficher, indiquant par des monticules de gravats les emplacements initiaux des ressources ayant étées détruites.

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	Mineur
Classe statique contenant que des méthodes statiques	Bao
Héritage	Gravats Terrain
Classe avec composition	Simulation
Classe avec un constructeur par copie ou clone()	Mineur
Noms des classes créées (entre 4 et 10 classes)	Bao Gravats Mineur Simulation TestSimulation

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

```
/** Classe Bao : Boite-à-outils
 * Classe statique contenant des méthodes statiques
 */
public class Bao {

   /** Constructeur vide permettant d'interdire l'instanciation de cette classe d'outils
   */
   private Bao(){
   }

   /** Methode permettant de fournir un nombre d'entier naturel compris entre le min et le max fournis
    *
    * @param min Correspond au minimum (inclu)
    * @param max Correspond au maximum (exclu)
    */
   public static int nbrAleatoire(int min,int max){
      return ((int) (Math.random() * (max - min) + min));
   }

   /** Permet de retourner True si le numero passé en premier argument est dans l'intervall former par les deux autres
    *
    * @param x Entier dont on cherche à determiner s'il se trouve dans l'intervalle donnée
    * @param a Borne inférieur de l'intervalle
    * @param b Borne supérieur de l'intervalle
    */
   public static boolean estEntre(int x, int a, int b){
      return ((a <= x) && (x <= b));
   }

   /** Permet de calculer la somme du contenu d'un tableau d'entier
    *
    * @param itab Tableau d'entier
    */
   public static int sommeTabEntier(int[] itab){
      int total =0;
      for (int i=0; i<itab.length; i++){
         total += itab[i];
      }
      return total;
   }

   /** Permet de calculer la distance, en nombre de déplacement, entre le village est une case du terrain
    * On décide de placer le village à la case (-1,-1), il se trouve donc à l'exterieur du terrain
    * @param x Correspond à l'abscisse de la case
    * @param y Correspond à l'ordonnée de la case
```

```
    */
   public static int distVillage(int x,int y){return 1+x+y;}

   /** Permet de calculer la distance la plus courte possible entre le village est une case du terrain(en nombre de
    * déplacement)
    * @param x Correspond à l'abscisse de la case
    * @param y Correspond à l'ordonnée de la case
    */
   public static int distVillageOpti(int x,int y){return (Math.min(x,y)+1);}

   /** Permet de calculer le nombre total de ressource à créer à partir du tableau d'initialisation de ressource
    *
    * @param stab Correspond au tableau d'initialisation des ressources (Tableau de String à deux dimensions)
    */
   public static int nbrRessourceTotal(String[][] stab){
      int total =0;
      for (int t=0; t<stab.length; t++){
         total += Integer.parseInt(stab[t][1]);
      }
      return total;
   }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/** Classe Gravats
 *
 * @param nom : nom de l'objet gravat
 * @param x : abscisse des gravats sur le terrain
 * @param y : ordonnée des gravats sur le terrain
 * @param q : quantité
 */
public class Gravats extends Ressource{
   private String nom = "Gravats"; // Les gravats n'ont pas besoin d'être dissossiable
   private int x ;
   private int y ;
   private final int q = 0; // final car on ne s'interesse pas à la quantité de gravats qu'on considére donc comme null

   /** Constructeur de Gravats
    */
   public Gravats(){
      super(" .°  . ",0);
   }

   /** Constructeur de Gravats avec coordonées explicites
    * Permet ainsi dès la création de l'objet gravat, de lui affecter des coordonnées précises
    * @param posx : abscisse à laquelle placer les gravats sur le terrain
```

```
     * @param posy : ordonnée à laquelle placer les gravats sur le terrain
     */
    public Gravats(int posx, int posy){
        this();
        this.x=posx;
        this.y=posy;
    }

    /** Constructeur de Gravats avec coordonées et terrain en paramètre ( pour le placement sur le terrain à la création!)
     * Permet ainsi dès la création de l'objet gravat, de le placer sur le terrain aux coordonnées passées en paramètres
     * si ces dernières sont sur le terrain.
     * @param posx : abscisse à laquelle placer les gravats sur le terrain
     * @param posy : ordonnée à laquelle placer les gravats sur le terrain
     * @param T : Terrain sur lequel placer l'objet
     */
    public Gravats(int posx, int posy, Terrain T){
        this(posx,posy);
        if(T.sontValides(posx,posy)){T.setCase(this.x,this.y,this);}
    }

    // GETTER SPETCIALE
    /** Retourne les informations de l'objet Gravats */
    public String toString(){
        return nom;
    }

    /** Retourne les informations de l'objet Gravats */
    public String phraseOcc(){
        return "Des "+nom+" encombre maintenant la case ("+x+":"+y+")";
    }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/** Classe Mineur
 *
 * @param nom : Correspond au nom du Mineur
 * @param px : abscisse initiale d'un Mineur à sa création
 * @param py : ordonnée initiale d'un Mineur à sa création
 * @param cpt : Compteur du nombre de Mineur crée, commence à -1 car le premier mineur est le chef du village!!
 * @param sac : Affiche le nombre de pépite d'or contenu dans le sac du mineur
 */
public class Mineur{
    // Tout Mineur à pour coordonnée de départ la case village (0:0)
    private String nom ;
    private int px ;
    private int py ;
    private static int cpt = -1; // static car à chaque création d'un mineur, Incrémentation de cette
```

```
variable.
   private int sac;

   /** Constructeur de Mineur.
    * Chaque mineur nés au village(point de coordonnée 0.0) ainsi à sa création, seul son nom est
demandé
    *
    * @param name : Correspond au nom du Mineur
    * @param posx : abscisse d'un Mineur
    * @param posy : ordonnée d'un Mineur
    * @param saccoche : Correspond au nombre de pépite d'or contenu dans le sac du mineur
    */
   public Mineur(String name,int posx,int posy,int saccoche){
      nom=name;
      px=posx;
      py=posy;
      sac=saccoche;
      cpt++; // Incrémentation du compteur de Mineur
   }

   /** Constructeur de Mineur par copie
    * Permet de créer une copie d'un mineur
    *
    * @param mineur : Correspond au nom du Mineur
    */
   public Mineur(Mineur autreMineur){
      this.nom = autreMineur.nom;
      this.px = autreMineur.px;
      this.py = autreMineur.py;
      this.sac = autreMineur.sac;
      cpt++; // Incrémentation du compteur de Mineur
   }

   // GETTER/SETTER
   /** Récupère le nom du Mineur */
   public String getName(){ return nom;}
   /** Modifie le nom du Mineur */
   public void setName(String newname){
      nom = newname;
   }
   /** Récupère la coordonnée d'abscisse de Mineur */
   public int getX(){ return px;}
   /** Récupère la coordonnée d'abscisse de Mineur */
   public int getY(){ return py;}
   /** Récupère la valeur de cpt : nombre de Mineur à créé */
   public static int getCpt(){ return cpt;}
   /** Récupère le contenue du sac du Mineur */
   public int getSac(){ return sac;}
   /** Modifie le contenu du sac du Mineur par l'entier passé en paramètre */
   public void setSac(int newsac){
```

```
        sac = newsac;
    }
    /** Retourne la nom et la position du Mineur sur le terrain */
    public String toString(){
        return "Mineur "+nom+" est dans la case ("+px+":"+py+") et a dans son sac "+sac+" pépites
d'Or.";
    }

    /** Déplace Mineur
     *
     * @param xnew Nouvelle abscisse de la position de Mineur
     * @param ynew Nouvelle ordonnée de la position de Mineur
     */
    public void seDeplacer(int xnew,int ynew){
        this.px=xnew;
        this.py=ynew;
    }

    /** Retourne la distance minimale séparant un mineur d'un objet présent sur le terrain
     *
     * @param x Nouvelle abscisse de la position de Mineur
     * @param y Nouvelle ordonnée de la position de Mineur
     */
    public int distance(int x,int y){
        int xlong = Math.abs(this.px - x);
        int ylong = Math.abs(this.py - y);
        int distance =(int) Math.sqrt((xlong)*(xlong) +(ylong)*(ylong));
        return distance;
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/** Lance les différentes phase composant la simulation:
 * <ul>
 * <li> Phase d'initialisation
 * <li> Phase de recherche, de transformation puis de récolte
 * </ul>
 *
 * @param t : Correspond au terrain sur lequel effectuer la simulation
 * @param tabInitRessource : Tableau contenant le nom et le nombre de ressources à ajouter au
terrain
 * @param tabNom : Tableau contenant les noms (et donc le nombre) des Mineur à créer
 * @param tabMineur : Tableau d'objet:Mineur
 * @param tabRessource : Tableau d'objet:Ressource
 * @param orTot : Variable contenant le nombre de pépites d'or total présentes sur le terrain.
 * @param nbr : Correspond au nombre de mineur à créé pour la simulation
 */
public class Simulation {

    private Terrain t;
    private String[][] tabInitRessource;
```

```
    private int orTot;
    private int nbr;
    private Mineur[] tabMineur;
    private Ressource[] tabRessource;
    private String[] tabNom={"Bob","Sam","Tom","Luck","Teo","Seb","Mat","JP","JD","PJ"};

    /** Constructeur de Simulation.
     *
     * @param ter1 : Correspond au terrain de la simulation
     * @param tab : Tableau de String associant les noms des ressources à créer ainsi que leurs
nombres
     * @param nbrMineur : Correspond au nombre de mineur choisit pour la simulation
     */
    public Simulation(Terrain ter1, String[][] tab,int nbrMineur){
        this.t=ter1;
        this.tabInitRessource=tab;
        this.tabNom=tabNom;
        this.nbr=nbrMineur;
    }

    // GETTEUR/SETTEUR
    /** Récupère la valeur de orTot : nombre de Mineur créé */
    public int getorTot(){ return orTot;}
    /** Redéfinie la valeur de orTot : nombre de Mineur créé */
    public void setorTot(int newVal){
        orTot = newVal;
    }

    /** Methode de lancement de la phase d'initialisation de l'environnement
     *
     * <li> Initialise les ressources via à la méthode initRandomRessources
     * <li> Initialise les Mineurs via la méthode initMineur
     */
    public void phaseInit(){
        System.out.println("PHASE 1:INITIALISATION DE L'ENVIRONNEMENT:\nInformations
sur les mineurs:");
        initMineur(this.nbr);                       // Initialisation des mineurs
        System.out.println("\nInformations sur les ressources:");
        initRandomRessources(this.tabInitRessource) ;   // Place aléatoirement les ressources sur le
terrain
        orTot=qTotRessource();                      // Calcul le nombre de pépites cachées et l'affecte à
orTot
        System.out.print("Il y a en tout "+orTot+" pépites d'or caché sur le terrain suivant : \n");
        System.out.println(t);                      // Affiche les infos du tableau
        t.affiche();                                // Affiche le terrain avec le ressources qu'il contient
    }

    /** Calcul la quantité totale de ressource disponible sur le terrain
    */
    private int qTotRessource(){
        int total=0;
```

```
        for (Ressource r: this.tabRessource){ total+=r.getQuantite(); }
        return total;
    }

    /** Initialisation des Mineurs, puis affiche une phrase indiquant le nombre de mineur actuellement créé
     * <ul>
     * <li> Créer le nombre de mineur choisi pour la simulation
     * <li> Leur affecte un nom à partir d'un tableau de nom
     * <li> Créer un tableau d'objet:Mineur
     * </ul>
     * @param arrayName Correspond au tableau contenant la liste des noms des mineurs
     */
    private void initMineur(int nbr){
        this.tabMineur = new Mineur[nbr];
        //Mineur initiale est le chef du village, il à donc pour coordonnée le village(-1,-1) et à un sac Vide
        //this.tabMineur[0] = new Mineur("Bob",-1,-1,0);
        Mineur m0 = new Mineur("Chef",-1,-1,0); // c'est à partir de lui que seront cloné les mineurs
        for (int i=0;i<nbr;i++){
            this.tabMineur[i] = new Mineur(m0);    // Créé un tableau de mineur
            this.tabMineur[i].setName(tabNom[i]);
            System.out.println(this.tabMineur[i].toString());
        }
        //// Affichage du nombre de Mineur créé
        //System.out.println("En tout, "+Mineur.getCpt()+" mineurs ont été créés.");
    }


    /** Affiche les objets Mineurs contenus dans le tableau d'objet tabMineur d'une Simulation
     */
    public void afficheTabMineur(){
        for (Mineur M: tabMineur){
            System.out.print(M.toString()+"\n");
        }
    }

    /** Permet, à partir d'un tableau de String de deux dimensions associant noms et nombres de ressources à créer, de
     * placer aléatoirement ces ressources sur le tableau
     *
     * @param tabR Correspond au tableau de String de deux dimensions permettant d'initialiser un type ainsi qu'un
     * nombre de ressource donné.Pour se faire il est composé de la manière suivante:
     * <ul>
     * <li> tabR[O][i] : correspondent aux noms des i-ressources.
     * <li> tabR[i][1] : correspondent aux quantités q des i-ressources.
     * </ul>
     */
    private void initRandomRessources(String[][] tabR){
```

```
      int taille = Bao.nbrRessourceTotal(tabR);         // Donne nombre d'éléments totaux du
tableaux.
      this.tabRessource = new Ressource[taille];        // Créer un tableau pouvant contenir tous les
éléments
      int cptR = 0;                                     // Compteur du nombre de ressource/objet créé(e)s
      for (int i =0; i<tabR.length; i++){
         String nom = tabR[i][0];                       // Récupération de la donnée du "nom" de la
ressource
         int m = Integer.parseInt(tabR[i][1]);          // Récupération de la donnée du "nbr" de
ressource(converti int)
         int cptBoucle = 0;                             // Compteur de boucle
         while (cptBoucle < m) {
            // Génération aléatoire des coordonnées et de la quantité des Arbres à créer
            int x = Bao.nbrAleatoire(0,t.nbLignes);   // Valeur aléatoire d'abscisse comprise entre 0
et longueur max terrain
            int y = Bao.nbrAleatoire(0,t.nbColonnes); // Valeur aléatoire d'ordonnée comprise entre
0 et largeur max terrain
            int q = Bao.nbrAleatoire(1,4);            // Valeur aléatoire de quantité comprise entre 1 et 3

            if (t.caseEstVide(x,y)){                  // Vérification si Vide, alors
               Ressource res1 = new Ressource(nom,q);// Crée la ressource
               t.setCase(x,y,res1);                   // La place sur le terrain
               this.tabRessource[cptR] = res1;        // L'Ajoute au tableau de ressource
               //System.out.println(" AJOUT :"+this.tabRessource[cptR].toString());
               cptR++;                                // incrémentation du compteur de ressource
               cptBoucle++ ;                          // incrémentation du compteur de boucle
            }
         }
         System.out.println("..."+m+" ressources "+nom+" ont été ajoutées...");
      }
   }

   /** Permet, pour un Mineur passé en paramètre, de faire les différents actions constituant la
récolte
    *
    * @param Min1 Correspond au mineur recoltant
    */
   private void recolte(Mineur Min1){
      int x = Min1.getX();
      int y = Min1.getY();
      Ressource R=t.getCase(x,y);                       // Mettre dans le sac du Mineur la ressource...
      if (Bao.estEntre(R.getQuantite(),2,20)){          // Si la quantité est comprise entre [1;20]
      System.out.println(Min1.getName()+" est tombé sur ("+R.toString()+", il pioche!");
      Min1.setSac(Min1.getSac() + 1 );                  // Remplis le sac du mineur avec la quantité d'or
trouvé
      R.setQuantite(R.getQuantite() - 1);               // Place la quantité de la ressource à zéro
      orTot--;                                          // Décrémente la var donnant le nombre d'or présent sur
terrain
      System.out.println(Min1.getName()+" à récolté 1 pépite d'or en ("+x+","+y+"), ce qui lui fait
```

```
un total de "+Min1.getSac()+" pépites d'or!");
      }else{                  // Quand c'est la dernière pépite alors...
         System.out.println(Min1.getName()+" est tombé sur ("+R.toString()+", il pioche!");
         Min1.setSac(Min1.getSac() + 1 );              // Remplis le sac du mineur avec la quantité d'or
trouvé
         R.setQuantite(R.getQuantite() - 1);          // Place la quantité de la ressource à zéro
         orTot--;                                     // Décrémente la var donnant le nombre d'or présent sur
terrain
         System.out.println(Min1.getName()+" à récolté 1 pépite d'or en ("+x+","+y+"), ce qui lui
fait un total de "+Min1.getSac()+" pépites d'or!");
         t.videCase(x,y);                             // Vider la case
         Gravats g1 = new Gravats(x,y,t);             // Place l'objet Gravas sur le tableau
         System.out.println("il n'y a plus rien ici...hormis quelques gravats!");
      }
   }

   /** Définie l'ensemble des actions possible pour un mineur pendant un tour de la phase de
recherche et recolte
    */
   public void recherchePiocheRecolte(){                // parcours dans l'ordre le tableau des Mineurs
      for (Mineur M: tabMineur){
         if (orTot<=0){                          // Stop le tour lorsq'un mineur autre que le dernier touve
la dernière pépite
            System.out.print("Désolé pour les autres mineurs de ce tour mais...");
            break;
         }else{
            int posx = M.getX();
            int posy = M.getY();
            System.out.println(M.getName()+" est en ("+posx+","+posy+")");
            if ( posx == -1 && posy == -1){          // S'IL SE TROUVE AU VILLAGE-->ENTRE
SUR LE TERRAIN EN 0:0
               System.out.println(M.getName()+" est au Village...il en sort!");
               M.seDeplacer(0,0);
               System.out.println(M.getName()+" s'est déplacé en ("+M.getX()+","+M.getY()+")");
            // S'IL EST SUR UNE CASE NON VIDE QUI N'EST PAS UN GRAVATS -->
RECOLTE
            }else if ((! t.caseEstVide(posx,posy))&&(t.getCase(posx,posy).toString()!="Gravats")){
               recolte(M);
            }
            else{            // S'IL EST SUR UNE CASE VIDE-->SE DEPLACE DE 1 CASE
               do{
                  int xalea = (Bao.nbrAleatoire(0,3)-1);
                  int yalea = (Bao.nbrAleatoire(0,3)-1);
                  if ((xalea != 0 || yalea != 0) && (t.sontValides(posx + xalea ,posy + yalea))){
                     M.seDeplacer(posx + xalea,posy + yalea);
                     System.out.println(M.getName()+" s'est déplacé en
("+M.getX()+","+M.getY()+")");
                  }
               }while(M.getX()==posx && M.getY()==posy);
               // PERMET DE FORCER LE DEPLACEMENT EN EXCLUANT DU TIRAGE
```

```
            } // LE SURPLACE!
          }
        }
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/**
 * @author Kenza BOUAICHA 3703308
 * @author Laurent GRONDIN 2767308
 */
public class TestSimulation {
    /**
     * @param args
     */
    public static void main(String[] args) {

// -[ JEUX DE DONNÉES UTILISABLES ]-----------------------------------------------------------------
-------------------
        // PETITE CONFIGURATION POUR LES TEST
        Terrain t0= new Terrain(5,5);
        String[][] ressource0={{"Roche","5"},{"Pierre","5"}};

        //// CONFIGURATION STANDART
        Terrain t= new Terrain(15,15);
        String[][] ressource={{"Granite","7"},{"Roche","7"},{"Cobalt","7"},{"Terre","7"}};
// -[ LANCEMENT DES SIMULATIONS ]-------------------------------------------------------------------
--------------------

        //lanceSimulationAvec(t0,ressource0,3);
        lanceSimulationAvec(t,ressource,6);
    }

    /** Lance une simulation avec le jeux de donnée passé en paramètre.
     *
     * @param t Correspond au terrain que l'on souhaite utiliser
     * @param ressource Correspond à la liste (String) permettant de définir les noms et les
quantitées d'objet
     * ressource à placer sur le terrain
     * @param nom est une liste de String contenant les noms des mineurs à placer dans la
simulation
     */
    private static void lanceSimulationAvec(Terrain ter1, String[][] ressource, int nbr ) {
        // PHASE 1 INITIALISATION
        Simulation s1 = new Simulation(ter1,ressource,nbr);
        s1.phaseInit();
        // PHASE 2 RECOLTE
        System.out.println("PHASE 2: RECHERCHE & RECOLTE");
        int tps = 0;                                    // Compteur de tour
        while (s1.getorTot()>0){                        // tant qu'il reste des pépites d'or sur le terrain
faire:
            System.out.print("-----\nTOUR "+tps+":\n");
```

```
            s1.rechercheChercheRecolte();
            tps++;                                  // incrémentation de la variable de tour
        }
        System.out.print("Il n'y a plus de pépites d'or disponible sur le terrain!\nFIN DE LA
RECOLTE\n\n");
        // PHASE 3 BILAN
        System.out.println("PHASE 3: BILAN");
        ter1.toString();                            // Affichage des infos du tableau
        ter1.affiche();                             // Affichage du tableau pour s'assurer qu'il soit vide
        System.out.print("Il a fallut "+tps+" tour pour récolter l'intégralité des pépites d'or:\n");
        s1.afficheTabMineur();
    }
}
```