Mini-projet LU2IN002 - 2020-2021

Nom : Abdallah	*Nom :*
Prénom : Sarah	*Prénom :*
N° étudiant : 3801002	*N° étudiant :*

Thème de simulation choisi (en 2 lignes max.)
Il s'agit de prisonniers (= agents) enfermés dans une cellule (= terrain) qui, pour être libres, doivent trouver les mots (= ressources) qui composent le code d'un cadenas.

Description des classes et de leur rôle dans la simulation (2 lignes max par classe)
La classe "Code" permet de mettre un terme à la simulation lorsque le nombre de mots trouvés par les prisonniers est supérieur ou égal au nombre de mots contenus dans le code. La classe "Prisonnier" permet le déplacement des prisonniers à l'aide de leurs coordonnées dans la cellule. La classe "Sirène" intervient lorsque la lettre trouvée est incorrecte ou lorsque les prisonniers ne parviennent pas à trouver le code dans le temps imparti.

Décrire, dans les grandes lignes, ce qui se passe durant la simulation (max. 5-6 lignes)
Durant la simulation, les prisonniers tentent de s'évader d'une cellule. Pour ce faire, ils se déplacent jusqu'aux coordonnées du cadenas pour deviner le code qui leur permettra de le déverrouiller. Si l'un des prisonniers parvient à trouver l'intégralité des mots que contient le code dans le temps imparti, les sirènes ne se déclenchent pas et un message s'affiche indiquant que les prisonniers ont réussi à s'échapper. Dans le cas contraire, la sirène retentit et un message indique que les prisonniers ne sont pas parvenus à s'échapper.

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	public class Prisonnier
Classe statique contenant que des méthodes statiques	public class Code
Héritage	public class Sirene
Classe avec composition	public classe Simulation

Classe avec un constructeur par copie ou clone()	public class Sirene
Noms des classes créées (entre 4 et 10 classes)	public class Simulation, public class Prisonnier, public class Code, public class Sirene.

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

```
public class Simulation{
   private int nb_lignes = 5;
   private int nb_colonnes = 6;
   private Ressource[] tabRessources; // Ce tableau contient les ressources.
   private Prisonniers[] tabPrisonniers; // Ce tableau contient les prisonniers.
   private Terrain cellule = new Terrain(nb_lignes, nb_colonnes);

   public Simulation(int a, int b){ // Initialisation du terrain avec les ressources.
      int i;
      int ligne = 0;
      int colonne = 0;
      tabRessources= new Ressource[a];
      tabPrisonniers = new Prisonnier[b];
      for (i = 0 ; i < a ; i++){
         tabRessources[i] = new Ressource("Lettre", (int)(Math.random()*3+1) ); // Les lettres
composant le code que les prisonniers doivent trouver pour pouvoir s'échapper.

         while (!(t.caseEstVide(ligne, colonne))){
            ligne = (int)(Math.random()*(nb_lignes));
            colonne = (int)(Math.random()*(nb_colonnes));
         }
         cellule.setCase(ligne, colonne, tabRessources[i]);
      }
      for(int i = 0 ; i < b ; i++){
         tabPrisonniers[i] = new Prisonnier(nb_lignes/2, nb_colonnes/2); // Les prisonniers
apparaissent à l'endroit où se situe le cadenas.
      }
      cellule.affiche();
   }

   public boolean pris(){
      int quantite;
      int aleatoire;
      Ressource res;
      Sirene s = new Sirene();
      for (Prisonnier p : tabPrisonniers){
         p.seDeplacer(); // Le prisonnier se déplace de façon aléatoire dans la cellule.
         if (cellule.getCase(p.getX(), p.getY()) != null){ // À condition que la cellule ne soit pas vide.
            aleatoire = (int)(Math.random()*100+1);
            if (aleatoire < 20){ // Il y'a 10% de chance que la lettre trouvée soit incorrecte et, par
conséquent, que la sirène se déclenche.
               cellule.videCase(p.getX(), p.getY());
               if (s.detecte()){
                  System.out.println("OuhouuuhOuhouuuh ! *La sirène retentit dans l'enceinte de la
cellule.*\n");
                  return false;
               }
            } else {
```

```
                res = cellule.getCase(p.getX(), p.getY());
                quantite = res.getQuantite();
                System.out.println("Le prisonnier est parvenu à trouver " + quantite + "lettres du
code.\n");

                while ((!p.mot_forme()) && (quantite != 0)){ // Le prisonnier mémorise les lettres.
                    p.ajouterLettre(new Ressource("Lettre", 1));
                    quantite --;
                    res.setQuantite(quantite);

                 }
                if (quantite == 0){
                    cellule.videCase(p.getX(), p.getY());
                } else {
                    cellule.setCase(p.getX(), p.getY(), res);}
            }
          }
       }
       return true;
   }
   public Prisonniers[] getTabPrisonniers(){
       return tabPrisonniers;
   }
   public int getNb_lignes(){
       return nb_lignes;
   }
   public int getNb_colonnes(){
       return nb_colonnes;
   }
}
```

```
public class Prisonnier{
        private x;
        private y;
        private Ressource[] tabLettres; // Ce tableau contient les lettres du code déjà trouvés par le
prisonnier.

        public Prisonnier(int x, int y){
                this.x = x;
                this.y = y;
                tabLettres = new Ressource[10];
        }
        public int distance(int x, int y){
                double m = (double) this.x-x;
                int n = this.y-y;
                return (int)Math.sqrt((m*m)+(n*n));
        }
        public void seDeplacer(int x, int y){
```

```
                this.x=x;
                this.y=y;
        }
        public void seDeplacer(){
                x = (int)(Math.random()*3) - 1 + x;
                y = (int)(Math.random()*3) - 1 + y;
        }
        public void ajouterLettre(Ressource lettre){ // Ajoute une lettre dans le tableau.
                int i = 0;
                if (this.tabLettres[0] == null){
                        this.tabLettres[i] = lettre;
                } else {
                        while ((this.tabLettres[i] != null) && (i < tabLettres.length)){
                                i++;
                        }
                        this.tabLettres[i] = lettre;
                }
        }
        public boolean mot_forme(){ // Retourne "True" si le prisonnier parvient à trouver 5 lettres
et, par conséquent, à former un mot du code.
                return tabLettres[4] != null;
        }
        public int getX(){
                return x;
        }
        public int getY(){
                return y;
        }
```

```
public class Code{
   int x;
   int y;
   int nb_mots;
   public Code(int x, int y){
      this.x = x;
      this.y = y;
      nb_mots = 0;
   }
   public int getX(){
      return this.x;
   }
   public int getY(){
      return this.y;
   }
   public int getNb_mots(){
      return nb_mots;
   }
   public void ajouter_mot(){
```

```
        nb_mots++;
    }
    public boolean codeTrouve(int nb_mots_du_code){
        return nb_mots >= nb_mots_du_code;
    }
}
```

```
public class Sirene extends Ressource{
        public Sirene(){ // Il s'agit du constructeur de la classe qui donne une case de la cellule contenant une sirène.
                super("Sirene", 1); // Chaque pièce continent 8 sirènes.
        }
        public boolean detecte(){ // Retourne "True" si la sirène détecte le prisonnier.
                int i = (int)(Math.random()*4+1);
                return i < 4;
        }
```