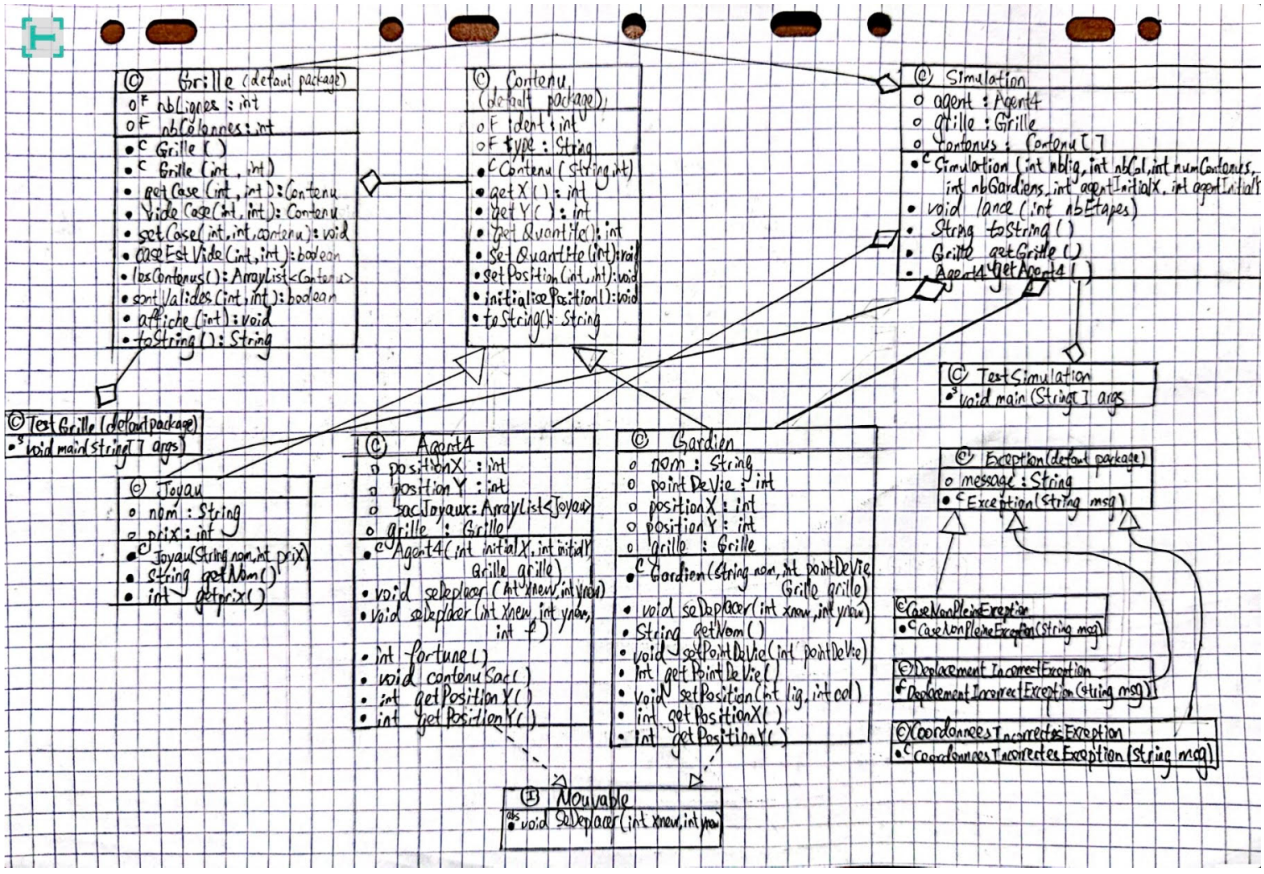
Projet LU2IN002 - 2023-2024

Numéro du groupe de TD/TME : 4

Nom : ZHANG	*Nom : LECOMTE*
Prénom : Yuxiang	*Prénom : Antoine*
N° étudiant : 21202829	*N° étudiant : 21103457*

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



Checklist des éléments utilisés:	*Nom(s) des classe(s) correspondante(s)*
Classe contenant un tableau ou une ArrayList	Simulation, Agent4
Classe avec membres et méthodes statiques	TestGrille, TestSimulation
Classe abstraite et méthode abstraite	(Pas utilisé)
Interface	Mouvable
Classe avec un constructeur par copie ou clone()	(Pas utilisé)

Définition de classe étendant Exception	DeplacementIncorrectException, CoordonneesIncorrectesException, CaseNonPleineException
Gestion des exceptions	Utilisation de throws et de son throw Utilisation de blocs try-catch
Utilisation du pattern singleton	(Pas utilisé)

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

```
/**
 * L'interface Mouvable définit le comportement des objets pouvant se déplacer dans la simulation.
 */
public interface Mouvable {
    /**
     * Déplace l'objet vers une nouvelle position spécifiée.
     *
     * @param xnew La nouvelle position en abscisse.
     * @param ynew La nouvelle position en ordonnée.
     * @throws DeplacementIncorrectException Si le déplacement est incorrect.
     */
    public void seDeplacer(int xnew, int ynew) throws DeplacementIncorrectException;
}

/**
 * Exception levée lorsqu'une opération qui nécessite une case non pleine est effectuée sur une case pleine.
 */
public class CaseNonPleineException extends Exception {
    /**
     * Constructeur de l'exception.
     * @param msg Message d'erreur associé à l'exception.
     */
    public CaseNonPleineException(String msg) {
        super(msg);
    }
}

/**
 * Exception levée lorsqu'une opération est effectuée avec des coordonnées incorrectes.
 */
public class CoordonneesIncorrectesException extends Exception {
    /**
     * Constructeur de l'exception.
```

```
     * @param msg Message d'erreur associé à l'exception.
     */
    public CoordonneesIncorrectesException(String msg) {
        super(msg);
    }
}

/**
 * Exception levée lorsqu'une tentative de déplacement est incorrecte.
 */
public class DeplacementIncorrectException extends Exception {
    /**
     * Constructeur de l'exception.
     * @param msg Message d'erreur associé à l'exception.
     */
    public DeplacementIncorrectException(String msg) {
        super(msg);
    }
}

/**
 * La classe Gardien représente un agent mobile de type Gardien dans la simulation.
 * Elle hérite de la classe Contenu et implémente l'interface Mouvable.
 */
public class Gardien extends Contenu implements Mouvable {
    private String nom;
    private int pointsDeVie;
    private int positionX;
    private int positionY;
    private Grille grille;

    /**
     * Constructeur de la classe Gardien.
     * @param nom Le nom du Gardien.
     * @param pointsDeVie Les points de vie du Gardien.
     * @param grille La grille sur laquelle le Gardien évolue.
     */
    public Gardien(String nom, int pointsDeVie, Grille grille) {
        super(nom, pointsDeVie);
        this.nom = nom;
        this.pointsDeVie = pointsDeVie;
        this.grille = grille;
        super.initialisePosition();
```

```
    }

    /**
     * Déplace le Gardien vers une nouvelle position spécifiée.
     * @param xnew La nouvelle coordonnée X.
     * @param ynew La nouvelle coordonnée Y.
     * @throws DeplacementIncorrectException Si le déplacement est incorrect.
     */
    public void seDeplacer(int xnew, int ynew) throws DeplacementIncorrectException {
        try {
            if (grille.sontValides(xnew, ynew)) {
                if (grille.caseEstVide(xnew, ynew)) {
                    grille.videCase(positionX, positionY);
                    positionX = xnew;
                    positionY = ynew;
                    grille.setCase(positionX, positionY, this);
                } else {
                    throw new DeplacementIncorrectException("Déplacement incorrect : la case n'est
pas vide");
                }
            } else {
                throw new DeplacementIncorrectException("Déplacement incorrect : coordonnées
invalides");
            }
        } catch (CoordonneesIncorrectesException | CaseNonPleineException e) {
            e.printStackTrace();
        }
    }

    /**
     * Obtient le nom du Gardien.
     * @return Le nom du Gardien.
     */
    public String getNom() {
        return nom;
    }

    /**
     * Obtient les points de vie du Gardien.
     * @return Les points de vie du Gardien.
     */
    public int getPointsDeVie() {
```

```
        return pointsDeVie;
    }

    /**
     * Modifie les points de vie du Gardien.
     * @param pointsDeVie Les nouveaux points de vie.
     */
    public void setPointsDeVie(int pointsDeVie) {
        this.pointsDeVie = pointsDeVie;
    }

    /**
     * Définit la position du Gardien sur la grille.
     * @param lig La ligne de la position.
     * @param col La colonne de la position.
     */
    public void setPosition(int lig, int col) {
        this.positionX = lig;
        this.positionY = col;
    }

    /**
     * Obtient la coordonnée X de la position du Gardien.
     * @return La coordonnée X de la position du Gardien.
     */
    public int getPositionX() {
        return positionX;
    }

    /**
     * Obtient la coordonnée Y de la position du Gardien.
     * @return La coordonnée Y de la position du Gardien.
     */
    public int getPositionY() {
        return positionY;
    }
}

/**
 * La classe Joyau représente un contenu de type Joyau dans la simulation.
 * Elle hérite de la classe Contenu.
 */
public class Joyau extends Contenu {
```

```
    private String nom;
    private int prix;

    /**
     * Constructeur de la classe Joyau.
     * @param nom Le nom du Joyau.
     * @param prix Le prix du Joyau.
    On ne peut pas utiliser les accesseurs ci-dessous si on n'ajoute pas les initialisation suivantes
    (l'appel au constructeur de la classe Contenu n'est pas suffisant).
    */
    public Joyau(String nom, int prix) {
        super(nom, prix);
        this.nom = nom;
        this.prix = prix;
    }

    /**
     * Obtient le nom du Joyau.
     * @return Le nom du Joyau.
     */
    public String getNom() {
        return nom;
    }

    /**
     * Obtient le prix du Joyau.
     * @return Le prix du Joyau.
     */
    public int getPrix() {
        return prix;
    }
}


/**
 * La classe Agent4 représente un agent mobile dans la simulation qui peut se déplacer sur une
grille
 * et interagir avec son contenu.
 */
import java.util.ArrayList;

public class Agent4 implements Mouvable {
    private int positionX;
```

```
    private int positionY;
    private ArrayList<Joyau> sacJoyaux;
    private Grille grille;

    /**
     * Constructeur de la classe Agent4.
     * @param initialX Position initiale en abscisse.
     * @param initialY Position initiale en ordonnée.
     * @param grille Grille sur laquelle l'agent évolue.
     */
    public Agent4(int initialX, int initialY, Grille grille) {
        this.positionX = initialX;
        this.positionY = initialY;
        this.grille = grille;
        this.sacJoyaux = new ArrayList<Joyau>();
    }

    /**
     * Permet à l'agent de se déplacer vers une nouvelle position spécifiée.
     * @param xnew Nouvelle position en abscisse.
     * @param ynew Nouvelle position en ordonnée.
     * @throws DeplacementIncorrectException Si le déplacement est incorrect.
     */
    public void seDeplacer(int xnew, int ynew) throws DeplacementIncorrectException {
        try {
            if (grille.sontValides(xnew, ynew)) {
                positionX = xnew;
                positionY = ynew;

                // Gérer le contenu de la case où l'agent arrive
                Contenu contenu = grille.getCase(positionX, positionY);
                if (contenu instanceof Joyau) {
                    sacJoyaux.add((Joyau) contenu);
                    grille.videCase(positionX, positionY);
                } else if (contenu instanceof Gardien) {
                    // Gérer le cas où un gardien est rencontré
                    System.out.println("Game Over: L'agent a rencontré un gardien sans force et a perdu
tous ses joyaux.");
                    sacJoyaux.clear();  // Vider le sac de joyaux
                }
            } else {
                throw new DeplacementIncorrectException("Déplacement incorrect");
```

```
        }
    } catch (CoordonneesIncorrectesException | CaseNonPleineException e) {
        e.printStackTrace();  // Ou toute autre forme de gestion d'erreur
    }
}

/**
 * Permet à l'agent de se déplacer vers une nouvelle position spécifiée avec une force donnée.
 * @param xnew Nouvelle position en abscisse.
 * @param ynew Nouvelle position en ordonnée.
 * @param f Force de l'agent.
 * @throws DeplacementIncorrectException Si le déplacement est incorrect.
 */
public void seDeplacer(int xnew, int ynew, int f) throws DeplacementIncorrectException {
    try {
        if (grille.sontValides(xnew, ynew)) {
            positionX = xnew;
            positionY = ynew;

            // Gérer le contenu de la case où l'agent arrive
            Contenu contenu = grille.getCase(positionX, positionY);
            if (contenu instanceof Joyau) {
                sacJoyaux.add((Joyau) contenu);
                grille.videCase(positionX, positionY);
            } else if (contenu instanceof Gardien) {
                Gardien gardien = (Gardien) contenu;
                if (gardien.getPointsDeVie() <= f) {
                    grille.videCase(positionX, positionY);
                } else {
                    System.out.println("Game Over: L'agent a rencontré un gardien et sa force n'est pas suffisante, et a perdu tous ses joyaux.");
                    sacJoyaux.clear();  // Vider le sac de joyaux
                    gardien.setPointsDeVie(gardien.getPointsDeVie() - f);
                }
            }
        } else {
            throw new DeplacementIncorrectException("Déplacement incorrect");
        }
    } catch (CoordonneesIncorrectesException | CaseNonPleineException e) {
        e.printStackTrace();  // Ou toute autre forme de gestion d'erreur
    }
}
```

```
    /**
     * Calcule et renvoie la fortune totale de l'agent en additionnant les prix de tous les joyaux dans son sac.
     * @return La fortune totale de l'agent.
     */
    public int fortune() {
        int totalPrix = 0;
        for (Joyau joyau : sacJoyaux) {
            totalPrix += joyau.getPrix();
        }
        return totalPrix;
    }

    /**
     * Affiche le contenu actuel du sac de l'agent en indiquant le nombre de chaque type de joyau.
     */
    public void contenuSac() {
        int nbDiamants = 0;
        int nbOpales = 0;
        int nbRubis = 0;

        if (sacJoyaux.isEmpty()) {
            System.out.println("Le sac est vide.");
        } else {
            System.out.println("Contenu du sac :");
            for (Joyau joyau : sacJoyaux) {
                if (joyau != null) {
                    if (joyau.getNom() != null) {
                        switch (joyau.getNom()) {
                            case "Diamant":
                                nbDiamants++;
                                break;
                            case "Opale":
                                nbOpales++;
                                break;
                            case "Rubis":
                                nbRubis++;
                                break;
                        }
                    }
                }
```

```
            }

            System.out.println("Nombre de Diamants : " + nbDiamants);
            System.out.println("Nombre d'Opales : " + nbOpales);
            System.out.println("Nombre de Rubis : " + nbRubis);
        }
    }

    /**
     * Renvoie la position en abscisse de l'agent.
     * @return La position en abscisse.
     */
    public int getPositionX() {
        return this.positionX;
    }

    /**
     * Renvoie la position en ordonnée de l'agent.
     * @return La position en ordonnée.
     */
    public int getPositionY() {
        return this.positionY;
    }
}

/**
 * La classe Simulation représente le déroulement de la simulation du jeu.
 */
public class Simulation {
    private Agent4 agent;
    private Grille grille;
    private Contenu[] contenus;

    /**
     * Initialise une nouvelle simulation avec les paramètres spécifiés.
     *
     * @param nbLig          Nombre de lignes de la grille.
     * @param nbCol          Nombre de colonnes de la grille.
     * @param numContenus    Nombre total de contenus à générer.
     * @param nbGardiens     Nombre de gardiens à générer.
     * @param agentInitialX  Position initiale en abscisse de l'agent.
     * @param agentInitialY  Position initiale en ordonnée de l'agent.
     */
```

```
public Simulation(int nbLig, int nbCol, int numContenus, int nbGardiens, int agentInitialX, int agentInitialY) {
    this.grille = new Grille(nbLig, nbCol);
    this.agent = new Agent4(agentInitialX, agentInitialY, this.grille);
    this.contenus = new Contenu[numContenus];

    if (numContenus > nbLig * nbCol) {
        System.out.println("Impossible de générer " + numContenus + " contenus dans une grille de " + nbLig + " lignes et " + nbCol + " colonnes.");
        return;
    }

    int contenusGeneres = 0;
    int gardiensGeneres = 0;
    int joyauxGeneres = 0;

    // Génération des gardiens
    while (gardiensGeneres < nbGardiens) {
        int x = (int) (Math.random() * nbLig);
        int y = (int) (Math.random() * nbCol);

        try {
            if (grille.caseEstVide(x, y)) {
                int pointsDeVieGardien = (int) (Math.random() * 201);
                Gardien gardien = new Gardien("Gardien", pointsDeVieGardien, this.grille);
                grille.setCase(x, y, gardien);
                contenus[contenusGeneres] = gardien;
                contenusGeneres++;
                gardiensGeneres++;
            }
        } catch (CoordonneesIncorrectesException e) {
            System.out.println(e.getMessage());
        }
    }

    // Génération des joyaux
    while (joyauxGeneres + gardiensGeneres < numContenus) {
        int x = (int) (Math.random() * nbLig);
        int y = (int) (Math.random() * nbCol);

        try {
            if (grille.caseEstVide(x, y)) {
                int prix = (int) (Math.random() * 4000 + 1);
```

```
                float randomType = (float) Math.random();

                Joyau joyau;
                try {
                    if (randomType < 1.0 / 3) {
                        joyau = new Joyau("Diamant", prix);
                    } else if (randomType < 2.0 / 3) {
                        joyau = new Joyau("Opale", prix);
                    } else {
                        joyau = new Joyau("Rubis", prix);
                    }

                    grille.setCase(x, y, joyau);
                    contenus[contenusGeneres] = joyau;
                    contenusGeneres++;
                    joyauxGeneres++;
                } catch (CoordonneesIncorrectesException e) {
                    System.out.println(e.getMessage());
                }
            }
        } catch (CoordonneesIncorrectesException e) {
            System.out.println(e.getMessage());
        }
    }
}

/**
 * Retourne une représentation textuelle de l'état actuel de la simulation.
 *
 * @return Une chaîne de caractères représentant l'état de la simulation.
 */
public String toString() {
    StringBuilder result = new StringBuilder();
    result.append("État de la simulation :\n");
    result.append("Grille :\n").append(grille).append("\n");
    result.append("Agent :\n").append("Position : (").append(agent.getPositionX()).append(",
").append(agent.getPositionY()).append(")\n");
    result.append("Fortune : ").append(agent.fortune()).append("\n");
    return result.toString();
}

/**
```

```
    * Lance la simulation pour un nombre spécifié d'étapes.
    *
    * @param nbEtapes Le nombre d'étapes de la simulation.
    */
   public void lance(int nbEtapes) {
      for (int i = 0; i < nbEtapes; i++) {
         System.out.println("Étape " + (i + 1) + " :\n");

         int xNew = agent.getPositionX() + (int)(Math.random() * 3 - 1);
         int yNew = agent.getPositionY() + (int)(Math.random() * 3 - 1);
         boolean avecForce = Math.random() < 0.3;
         int force = avecForce ? (int)(Math.random() * 91 + 10) : 0;

         try {
            if (avecForce) {
               agent.seDeplacer(xNew, yNew, force);
            } else {
               agent.seDeplacer(xNew, yNew);
            }
         } catch (DeplacementIncorrectException e) {
            System.out.println("Erreur de déplacement : " + e.getMessage());
         }

         // Déplacement des gardiens
         for (Contenu contenu : contenus) {
            if (contenu instanceof Gardien) {
               Gardien gardien = (Gardien) contenu;
               int newX = gardien.getPositionX() + (int)(Math.random() * 3 - 1);
               int newY = gardien.getPositionY() + (int)(Math.random() * 3 - 1);

               try {
                  gardien.seDeplacer(newX, newY);
               } catch (DeplacementIncorrectException ignored) {
                  // Pour ne pas rendre illisible l'affichage à l'exécution, on n'affiche rien pour
l'exception.
               }
            }
         }

         System.out.println("Informations sur l'étape :\n");
         System.out.println(this);
```

```
            try {
                Thread.sleep(1000); // attendre 1 seconde pour avoir le temps de voir les changements
dans la grille à l'exécution.
            } catch (InterruptedException e) {
                e.getMessage();
            }

            grille.affiche(10); // ajuster la taille pour l'affichage de la grille
        }
    }

    /**
     * Retourne la grille de la simulation.
     *
     * @return La grille de la simulation.
     */
    public Grille getGrille(){
        return grille;
    }

    /**
     * Retourne l'agent de la simulation.
     *
     * @return L'agent de la simulation.
     */
    public Agent4 getAgent4(){
        return agent;
    }
}

 /**
 * La classe TestSimulation est utilisée pour tester la simulation du jeu.
 */
public class TestSimulation {
    public static void main(String[] args) {
        // Créer une instance de Simulation avec des paramètres appropriés
        Simulation simulation = new Simulation(10, 10, 20, 3, 0, 0);

        // Exécuter la simulation avec un nombre spécifié d'étapes
        simulation.lance(20);

        // Récupérer la grille et l'agent de la simulation
        Grille grille = simulation.getGrille();
```

```
        Agent4 agent = simulation.getAgent4();

        // Afficher les informations finales de la simulation
        System.out.println("Simulation terminée. Informations finales :\n");
        System.out.println("Agent position : (" + agent.getPositionX() + ", " + agent.getPositionY()
+ ")");
        System.out.println("Agent fortune : " + agent.fortune() + " pièces d'or");
        agent.contenuSac();

        // Afficher le contenu final de la grille
        System.out.println("\nContenu final de la grille :\n" + grille);
    }
}
```