DUT Informatique, Paris DESCARTES

1e année

Projet Java

MULOTS



Antoine WANG (Groupe 107)

Osman KANTAR (Groupe 101)

Table des matières

[Introduction 3](#_Toc452117878)

[Diagramme uml 4](#_Toc452117879)

[Test unitaires 5](#_Toc452117880)

[Sources 6](#_Toc452117881)

[Bilan 7](#_Toc452117882)

# Introduction

Il nous ait demandé de développer un jeu de mulots en java et d’y faire des tests unitaires. Les mulots partent tous en chaîne, les uns après les autres, d’une entrée E et le joueur doit leur attribuer des métiers pour tous les faire atteindre la sortie S.

Ces déplacements sont visibles dans la console grâce à une boucle les deux emplacements « Entrée et Sortie » ont chacun leurs coordonnées dans un tableau que nous appellerons Niveau.

Pour interagir avec les mulots marcheurs, c’est-à-dire les mulots basiques, qui sont numérotés de 0 à X selon le niveau, le joueur entre le numéro suivi de la lettre du métier en question.

Dans notre cas, nous avons le parachutiste = P

Ou le grimpeur = G

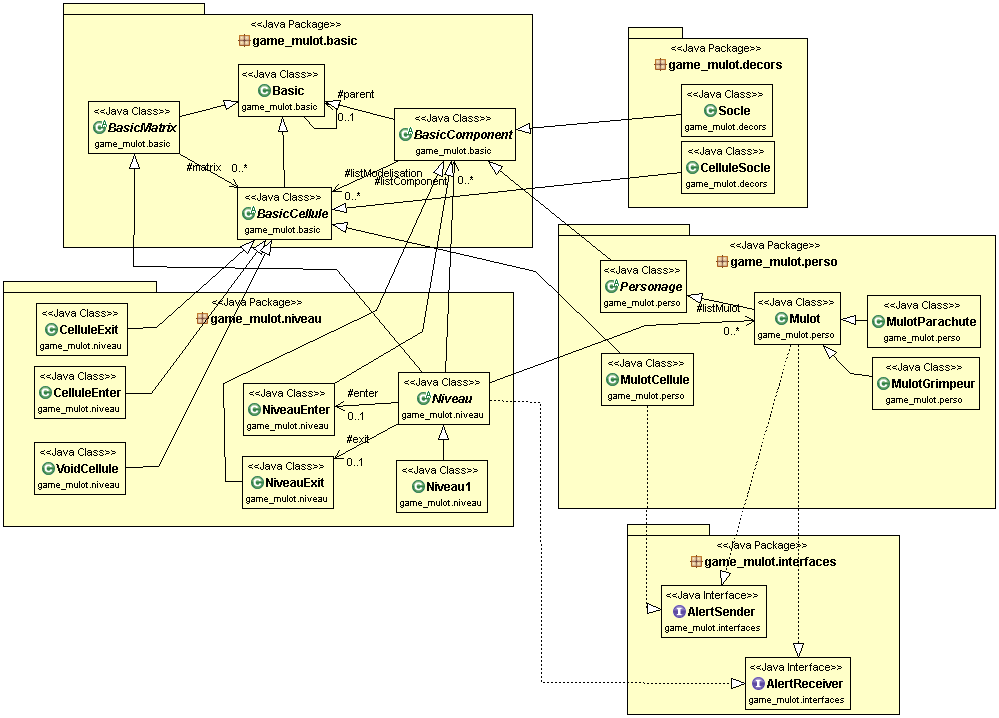
Ces métiers sont affichés durant la partie, ainsi que le nombre restant de leur utilisation.

Un mulot qui tombe de plus de 5 cases sans être parachutiste meurt, un mulot qui sort hors des limites du niveau est aussi considéré comme mort.

Dans notre code, le déplacement des mulots est surtout basé sur notre boolean crossable, en réalité, en vérifiant le type de case qui se trouve à la case d’après, le mulot tombera ou effectuera un demi-tour. S’il s’agit d’une case vide, alors c’est « crossable » donc le mulot continuera son avancée.

De plus, notre code reste très ouvert pour un codage à l’avenir, car en se basant simplement sur ce boolean, nous aurions pu coder les creuseurs et les perceurs, en leur donnant simplement des conditions, les murs et les sols seraient devenus crossable pour ces métiers-là.

# Diagramme uml



# Test unitaires

import static org.junit.Assert.\*;

import java.util.ArrayList;

import org.junit.Test;

import game\_mulot.basic.BasicComponent;

import game\_mulot.decors.Socle;

import game\_mulot.niveau.Niveau1;

public class TestsUnitaires {

@Test

public void test() {

ArrayList<BasicComponent> list1 = new ArrayList<BasicComponent>();

list1.add(new Socle(55, 20, 15, Socle.SocleOrientation.HORIZONTAL, null));

Niveau1 n1 = new Niveau1(list1, null);

n1.draw();

n1.tic(); // Ajoute mulot

assertEquals(10, n1.getEnter().getPosX());

assertEquals(0, n1.getEnter().getPosY());

assertEquals(69, n1.getExit().getPosX());

assertEquals(19, n1.getExit().getPosY());

assertNotNull("La liste de mulot n'est pas nul", n1.getListMulot());

}

}

# Sources

Basic

package game\_mulot.basic;

/\*\*

\* <p>

\* Cette class est la class basic dont tous les objets hérites. Elle est

\* composée d'une variable <strong>name</strong> qui lui permet d'avoir un nom

\* et aussi d'une variable Basic <strong>parent</strong> qui permet de récupérer

\* son parent qui lui même aura un parent, on peut donc grâce à ce système avoir

\* accès à des objets facilement comme le principe d'une liste chaînée.

\* </p>

\*

\*

\*/

public class Basic {

// Variables.

/\*\*

\* Le nom du Basic.

\*/

protected String name;

/\*\*

\* Le parent du Basic.

\*/

protected Basic parent;

// Constructors.

/\*\*

\* Initialise le nom à "" et le parent null.

\*/

public Basic() {

this.name = "";

this.parent = null;

}

/\*\*

\* Donne un nom est un parent au Basic.

\*

\* @param name

\* - le nom du Basic.

\* @param parent

\* - le parent du Basic.

\*/

public Basic(String name, Basic parent) {

this.name = name;

this.parent = parent;

}

// Public methods.

@Override

public String toString() {

return "Basic [name=" + name + ", parent=" + ((parent == null) ? "NULL" : parent) + "]";

}

// Getters and Setters.

/\*\*

\* @return the name

\*/

public String getName() {

return name;

}

/\*\*

\* @param name

\* the name to set

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return the parent

\*/

public Basic getParent() {

return parent;

}

/\*\*

\* @param parent

\* the parent to set

\*/

public void setParent(Basic parent) {

this.parent = parent;

}

}

BasicCellule

package game\_mulot.basic;

import java.awt.Color;

import java.awt.Graphics;

/\*\*

\* <p>

\* Class qui représente une cellule d'un GraphicObject. Un GraphicObject peut

\* donc contenir une ou plusiseurs BasicCellule.

\* </p>

\* <p>

\* Une BasicCellule possède une coordonnée X et une coordonnée Y représentant sa

\* position (A vous de voir si la position est celle dans l'objet dans laquell

\* elle est ou si c'est dans un autre objet).

\* </p>

\* <p>

\* Une basicCellule possède aussi un Objet parent qui permet de savoir à qui

\* elle est raliée. (ce parent peut-être null).

\* </p>

\* <p>

\* Elle possède aussi un boolean disant si cette cellule est traversable ou non,

\* cela permet de vérifier lors de déplacements si l'on peut se trouver ou non à

\* la même place qu'elle.

\* </p>

\* <p>

\* Enfin elle possède une couleur et un objet qui sera son apparence. Il faudra

\* alors ensuite implementer les méthodes <strong>draw(Graphic g)</strong> et

\* <strong>draw()</strong> pour définir comment utiliser comme vous le voulez

\* ces propriétées.

\* </p>

\*

\* <p>

\* <strong>Liste des propriétés:</br>

\* </strong>

\* <ul>

\* <li>posX</li>

\* <li>posY</li>

\* <li>crossable</li>

\* <li>parent</li>

\* <li>color</li>

\* <li>apparence</li>

\* </ul>

\* </p>

\*

\*

\*/

public abstract class BasicCellule extends Basic {

// Variables.

/\*\*

\* La coordonnée X de la cellule.

\*/

protected int posX;

/\*\*

\* La coordonnée Y de la cellule.

\*/

protected int posY;

/\*\*

\* Le boolean pour savoir si la cellule peut être traversée ou non.

\*/

protected boolean crossable;

/\*\*

\* La couleur de la cellule.

\*/

protected Color color;

/\*\*

\* L'objet qui peut représenter la cellule.

\*/

protected Object apparence;

// Constructors.

/\*\*

\* Initialise toutes les propriétés à zéro, true ou null.

\*/

public BasicCellule() {

super();

this.posX = 0;

this.posY = 0;

this.crossable = true;

this.parent = null;

this.color = null;

this.apparence = null;

}

/\*\*

\* <p>

\* Donne des positions X et Y à la cellule, un état traversable ou non et un

\* parent.

\* </p>

\*

\* @param posX

\* - la position X de la cellule.

\* @param posY

\* - la position Y de la cellule.

\* @param crossable

\* - l'état traversable ou non de la cellule.

\* @param parent

\* - le parent de la celllule.

\*/

public BasicCellule(int posX, int posY, boolean crossable, Basic parent) {

super("Cellule", parent);

this.setPosX(posX);

this.setPosY(posY);

this.setCrossable(crossable);

this.setColor(null);

this.setApparence(null);

}

/\*\*

\* <p>

\* Permet d'initialiser toutes les propriété de la cellule.

\* </p>

\*

\* @param posX

\* - la position X.

\* @param posY

\* - la position Y.

\* @param crossable

\* - le boolean pour savoir si elle est traversable ou non.

\* @param color

\* - la couleur.

\* @param apparence

\* - l'apparence.

\* @param parent

\* - le parent.

\*/

public BasicCellule(int posX, int posY, boolean crossable, Color color, Object apparence, Basic parent) {

super("Cellule", parent);

this.setPosX(posX);

this.setPosY(posY);

this.setCrossable(crossable);

this.setColor(color);

this.setApparence(apparence);

}

// Abstract methods.

/\*\*

\* Permet de dessiner la cellule avec un Graphics.

\*

\* @param g

\* - l'objet qui va dessiner l'objet.

\*/

public abstract void draw(Graphics g);

/\*\*

\* Permet de dessiner l'objet.

\*/

public abstract void draw();

/\*\*

\* <p>

\* Vérifie la position indiquée en paramètre pour savoir si la cellule peut

\* être déplacer à cette position ou non.

\* </p>

\*

\* <strong>Utilisé dans les setters de posX et posY.</strong>

\*

\* @param posX

\* - la coordonnée x de la position à vérifier.

\* @param posY

\* - la coordonnée y de la position à vérifier.

\*

\* @return true si la cellule peut être déplacer aux positions indiquées

\* sinon false.

\*/

public abstract boolean verifyPosition(int posX, int posY);

// Public methods.

@Override

public String toString() {

return "BasicCellule [posX=" + posX + ", posY=" + posY + ", crossable=" + crossable + ", parent="

+ ((parent == null) ? "NULL" : parent.toString()) + ", color="

+ ((color == null) ? "NULL" : color.toString()) + ", apparence="

+ ((apparence == null) ? "NULL" : apparence.toString()) + "]";

}

// Getters and Setters.

/\*\*

\* @return the posX

\*/

public int getPosX() {

return posX;

}

/\*\*

\* Change de position X

\*

\* @param posX

\* the posX to set

\*/

public void setPosX(int posX) {

this.posX = posX;

}

/\*\*

\* @return the posY

\*/

public int getPosY() {

return posY;

}

/\*\*

\* Change de position Y.

\*

\* @param posY

\* the posY to set

\*/

public void setPosY(int posY) {

this.posY = posY;

}

/\*\*

\* @return the crossable

\*/

public boolean isCrossable() {

return crossable;

}

/\*\*

\* @param crossable

\* the crossable to set

\*/

public void setCrossable(boolean crossable) {

this.crossable = crossable;

}

/\*\*

\* @return the color

\*/

public Color getColor() {

return color;

}

/\*\*

\* @param color

\* the color to set

\*/

public void setColor(Color color) {

this.color = color;

}

/\*\*

\* @return the apparence

\*/

public Object getApparence() {

return apparence;

}

/\*\*

\* @param apparence

\* the apparence to set

\*/

public void setApparence(Object apparence) {

this.apparence = apparence;

}

}

BasicComponent

package game\_mulot.basic;

import java.awt.Graphics;

import java.util.ArrayList;

/\*\*

\* <p>

\* Objet composé de BasicCellule dans un tableau. Il possède une position X etun

\* position Y et un Objet parent.

\* </p>

\*

\* <p>

\* <strong>Liste des propriétés :</br>

\* </strong>

\*

\* <ul>

\* <li>name</li>

\* <li>posX</li>

\* <li>posY</li>

\* <li>tabModelisation</li>

\* <li>parent</li>

\* </ul>

\* </p>

\*

\*/

public abstract class BasicComponent extends Basic {

// Variables.

/\*\*

\* La position X de l'objet.

\*/

protected int posX;

/\*\*

\* La position Y de l'objet.

\*/

protected int posY;

/\*\*

\* La liste qui contient toutes les cellules qui vont modéliser l'objet.

\*/

protected ArrayList<BasicCellule> listModelisation;

// Constructors.

/\*\*

\* Initialise à 0, "" ou null toutes les propriétés.

\*/

public BasicComponent() {

super();

this.initializeListModelisation();

this.posX = 0;

this.posY = 0;

this.listModelisation = null;

}

public BasicComponent(String name, int posX, int posY, Basic parent) {

super(name, parent);

this.initializeListModelisation();

this.setPosX(posX);

this.setPosY(posY);

}

// Abstract methods.

/\*\*

\* Initialise le tableau de BasicCellule.

\*/

protected abstract void initializeListModelisation();

/\*\*

\* Dessine l'objet avec un Objet Graphics.

\*

\* @param g

\* - objet Graphic qui va dessiner l'objet.

\*/

public abstract void draw(Graphics g);

/\*\*

\* Dessine l'objet.

\*/

public abstract void draw();

/\*\*

\* Permet de vérifier une position pour savoir si l'objet peut se déplace à

\* cette position ou non. (Méthode utilisée dans les setters de posX et

\* posY).

\*

\* @param posX

\* - la coordonnée X de la position à vérifier.

\* @param posY

\* - la coordonnée Y de la position à vérifier.

\*

\* @see BasicComponent#setPosX(int)

\* @see BasicComponent#setPosY(int)

\*

\* @return true si l'objet peut se déplacer à la position sinon false.

\*/

public abstract boolean verifyPosition(int posX, int posY);

// Public methods.

@Override

public String toString() {

return "BasicObject [name=" + name + ", posX=" + posX + ", posY=" + posY + ", parent="

+ ((parent == null) ? "NULL" : parent.toString()) + "]";

}

// Getters and Setters.

/\*\*

\* @return the posX

\*/

public int getPosX() {

return posX;

}

/\*\*

\* Change de position X si la nouvelle position a été vérifiée est

\* approuvée.

\*

\* @param posX

\* the posX to set

\*

\* @see BasicCellule#verifyPosition(int, int)

\*/

public void setPosX(int posX) {

// On vérifie si on peut se déplacer.

if (this.verifyPosition(posX, this.getPosY())) {

// On fait bouger toutes les cellules.

for (BasicCellule cellule : this.listModelisation) {

cellule.setPosX(posX - (this.getPosX() - cellule.getPosX()));

}

// On change la position.

this.posX = posX;

}

}

/\*\*

\* @return the posY

\*/

public int getPosY() {

return posY;

}

/\*\*

\* Change de position Y si la nouvelle position a été vérifiée est

\* approuvée.

\*

\* @param posY

\* the posY to set

\*

\* @see BasicCellule#verifyPosition(int, int)

\*/

public void setPosY(int posY) {

// On vérufue si on peut se déplacer.

if (this.verifyPosition(this.getPosX(), posY)) {

// On fait bouger toutes les cellules.

for (BasicCellule cellule : this.listModelisation) {

cellule.setPosY(posY - (this.getPosY() - cellule.getPosY()));

}

// On change la position.

this.posY = posY;

}

}

/\*\*

\* @return the tabModelisation

\*/

public ArrayList<BasicCellule> getListModelisation() {

return listModelisation;

}

}

BasicMatrix

package game\_mulot.basic;

import java.awt.Graphics;

import java.util.ArrayList;

/\*\*

\* <p>

\* Matrix de basic cellule qui peut représenter une map par exemple.

\* </p>

\* <p>

\* Cette objet possède comme propriétés le nombre de line, le nombre de

\* colonnes, le tableau de Basic à une dimension qui représente la matrice et la

\* liste des composants à afficher dans la matrice.

\* </p>

\*

\*

\*/

public abstract class BasicMatrix extends Basic {

// Variables.

/\*\*

\* Le nombre de lignes de la matrice.

\*/

protected int lines;

/\*\*

\* Le nombre de colonnes de la matrice.

\*/

protected int columns;

/\*\*

\* Le tableau de cellules qui représente la matrice.

\*/

protected BasicCellule matrix[];

// Constructors

public BasicMatrix() {

super();

this.lines = 1;

this.columns = 1;

this.initializeMatrix();

}

public BasicMatrix(String name, int lines, int columns, ArrayList<BasicComponent> listComponent, Basic parent) {

super(name, parent);

if (lines >= 1)

this.lines = lines;

else

this.lines = 1;

if (columns >= 1)

this.columns = columns;

else

this.columns = 1;

this.initializeMatrix();

}

// Abstract methods.

/\*\*

\* <p>

\* Permet d'initialiser la matrice de BasicCellules. (Cette méthode est

\* utilisée dans le constructeur de la class <strong>BasicObject</strong>)

\* </p>

\*

\* @see BasicMatrix#BasicMatrix()

\* @see BasicMatrix#BasicMatrix(String, int, int, ArrayList, Basic)

\* @see BasicMatrix

\*/

protected abstract void initializeMatrix();

/\*\*

\* Permet d'inclure les composants dans la matrices.

\*/

protected abstract void includeComponentInMatrix();

/\*\*

\* Permet de dessiner la matrix à partir d'un objet Graphics.

\*

\* @param g

\* - Graphic qui va dessiner la BasicMatrix.

\*/

public abstract void draw(Graphics g);

/\*\*

\* Permet de dessiner la matrice.

\*/

public abstract void draw();

// Public methods.

@Override

public String toString() {

return "BasicMatrix [name=" + name + ", lines=" + lines + ", columns=" + columns + ",parent=" + parent + "]";

}

// Getters and Setters.

/\*\*

\* @return the lines

\*/

public int getLines() {

return lines;

}

/\*\*

\* @param lines

\* the lines to set

\*/

public void setLines(int lines) {

if (lines >= 1) {

this.lines = lines;

this.initializeMatrix();

}

}

/\*\*

\* @return the columns

\*/

public int getColumns() {

return columns;

}

/\*\*

\* @param columns

\* the columns to set

\*/

public void setColumns(int columns) {

if (columns >= 1) {

this.columns = columns;

this.initializeMatrix();

}

}

/\*\*

\* @return the matrix

\*/

public BasicCellule[] getMatrix() {

return matrix;

}

}

Outils

package game\_mulot.basic;

import java.awt.Graphics;

import java.util.ArrayList;

/\*\*

\* <p>

\* Matrix de basic cellule qui peut représenter une map par exemple.

\* </p>

\* <p>

\* Cette objet possède comme propriétés le nombre de line, le nombre de

\* colonnes, le tableau de Basic à une dimension qui représente la matrice et la

\* liste des composants à afficher dans la matrice.

\* </p>

\*

\*

\*/

public abstract class BasicMatrix extends Basic {

// Variables.

/\*\*

\* Le nombre de lignes de la matrice.

\*/

protected int lines;

/\*\*

\* Le nombre de colonnes de la matrice.

\*/

protected int columns;

/\*\*

\* Le tableau de cellules qui représente la matrice.

\*/

protected BasicCellule matrix[];

// Constructors

public BasicMatrix() {

super();

this.lines = 1;

this.columns = 1;

this.initializeMatrix();

}

public BasicMatrix(String name, int lines, int columns, ArrayList<BasicComponent> listComponent, Basic parent) {

super(name, parent);

if (lines >= 1)

this.lines = lines;

else

this.lines = 1;

if (columns >= 1)

this.columns = columns;

else

this.columns = 1;

this.initializeMatrix();

}

// Abstract methods.

/\*\*

\* <p>

\* Permet d'initialiser la matrice de BasicCellules. (Cette méthode est

\* utilisée dans le constructeur de la class <strong>BasicObject</strong>)

\* </p>

\*

\* @see BasicMatrix#BasicMatrix()

\* @see BasicMatrix#BasicMatrix(String, int, int, ArrayList, Basic)

\* @see BasicMatrix

\*/

protected abstract void initializeMatrix();

/\*\*

\* Permet d'inclure les composants dans la matrices.

\*/

protected abstract void includeComponentInMatrix();

/\*\*

\* Permet de dessiner la matrix à partir d'un objet Graphics.

\*

\* @param g

\* - Graphic qui va dessiner la BasicMatrix.

\*/

public abstract void draw(Graphics g);

/\*\*

\* Permet de dessiner la matrice.

\*/

public abstract void draw();

// Public methods.

@Override

public String toString() {

return "BasicMatrix [name=" + name + ", lines=" + lines + ", columns=" + columns + ",parent=" + parent + "]";

}

// Getters and Setters.

/\*\*

\* @return the lines

\*/

public int getLines() {

return lines;

}

/\*\*

\* @param lines

\* the lines to set

\*/

public void setLines(int lines) {

if (lines >= 1) {

this.lines = lines;

this.initializeMatrix();

}

}

/\*\*

\* @return the columns

\*/

public int getColumns() {

return columns;

}

/\*\*

\* @param columns

\* the columns to set

\*/

public void setColumns(int columns) {

if (columns >= 1) {

this.columns = columns;

this.initializeMatrix();

}

}

/\*\*

\* @return the matrix

\*/

public BasicCellule[] getMatrix() {

return matrix;

}

}

CelluleSocle

package game\_mulot.decors;

import java.awt.Color;

import java.awt.Graphics;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.niveau.Niveau;

import game\_mulot.niveau.VoidCellule;

/\*\*

\* <p>

\* La cellule qui va composer un <strong>Socle</strong>. Elle n'est pas

\* traversable et est par défaut représenter par un Character('-').

\* </p>

\*

\*

\*/

public class CelluleSocle extends BasicCellule {

public CelluleSocle() {

super();

this.crossable = false;

this.apparence = new Character('-');

}

public CelluleSocle(int posX, int posY, Basic parent) {

super(posX, posY, true, parent);

this.crossable = false;

this.apparence = new Character('-');

}

public CelluleSocle(int posX, int posY, Color color, Object apparence, Basic parent) {

super(posX, posY, false, color, apparence, parent);

}

@Override

public void draw(Graphics g) {

// Non utilisé.

}

@Override

public void draw() {

System.out.print(this.getApparence());

}

@Override

public boolean verifyPosition(int posX, int posY) {

if (this.parent != null) { // Si on a pas un parent null.

if (this.parent.getParent() != null) {

if (this.parent.getParent().getClass() == Niveau.class) {

Niveau n = ((Niveau) this.parent.getParent());

// Si la case où l'on veut aller est une VoidCellule.

if (n.getMatrix()[posY \* n.getColumns() + posY].getClass() == VoidCellule.class) {

return true;

} else {

return false;

}

} else {

return true;

}

} else {

return true;

}

} else {

return true;

}

}

}

Socle

package game\_mulot.decors;

import java.awt.Graphics;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicComponent;

/\*\*

\* <p>

\* Objet qui représente un socle soit horizontal, soit vertical. Ses positions X

\* et Y sont celle de la première cellule à gauche dans le cas où le socle est

\* Horizontal, sinon la cellule en haut. Le socle possède aussi une longueur qui

\* n'a pas d'unité (peut-être en case de tableau ou en pixel ou autre).

\* </p>

\*

\*

\*/

public class Socle extends BasicComponent {

// Variables.

/\*\*

\* La longueur du Socle

\*/

protected int length;

/\*\*

\* L'orientation du socle.

\*/

protected Socle.SocleOrientation orientation;

// Constructors.

public Socle() {

super();

this.name = "Socle";

this.length = 1;

this.orientation = Socle.SocleOrientation.HORIZONTAL;

// On re initialise le tableau.

this.initializeListModelisation();

}

public Socle(int posX, int posY, int length, Socle.SocleOrientation orientation, Basic Parent) {

super("Socle", posX, posY, Parent);

if (length >= 1) {

this.length = length;

} else {

this.length = 1;

}

if (orientation != null) {

this.orientation = orientation;

} else {

this.orientation = Socle.SocleOrientation.HORIZONTAL;

}

// On re initialise le tableau.

this.initializeListModelisation();

}

// Inherited methods.

@Override

protected void initializeListModelisation() {

if (this.orientation == Socle.SocleOrientation.HORIZONTAL) {

this.listModelisation = new ArrayList<BasicCellule>();

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

this.listModelisation

.add(new CelluleSocle(this.getPosX() + i, this.getPosY(), null, new Character('-'), this));

}

} else {

this.listModelisation = new ArrayList<BasicCellule>();

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

this.listModelisation

.add(new CelluleSocle(this.getPosX(), this.getPosY() + i, null, new Character('|'), this));

}

}

}

@Override

public void draw(Graphics g) {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw(g);

}

}

@Override

public void draw() {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw();

}

}

@Override

public boolean verifyPosition(int posX, int posY) {

boolean verif = false;

if (this.getListModelisation() != null) {

if (!this.getListModelisation().isEmpty()) {

// On vérifie si toutes les cellule peuvent être placée à leur

// endroit respective.

for (BasicCellule cellule : this.getListModelisation()) {

verif = cellule.verifyPosition(posX - (this.getPosX() - cellule.getPosX()),

posY - (this.getPosY() - cellule.getPosY()));

if (!verif) {

break;

}

}

} else {

verif = true;

}

} else {

verif = true;

}

return verif;

}

// Getters and Setters.

/\*\*

\* @return the length

\*/

public int getLength() {

return length;

}

/\*\*

\* @param length

\* the length to set

\*/

public void setLength(int length) {

if (length >= 1) {

this.length = length;

this.initializeListModelisation();

}

}

/\*\*

\* @return the orientation

\*/

public Socle.SocleOrientation getOrientation() {

return orientation;

}

/\*\*

\* @param orientation

\* the orientation to set

\*/

public void setOrientation(Socle.SocleOrientation orientation) {

if (this.orientation != null) {

this.orientation = orientation;

this.initializeListModelisation();

}

}

// Enumerations.

public enum SocleOrientation {

HORIZONTAL("HORIZONTAL"), VERTICAL("VERTICAl");

private String text = "";

SocleOrientation(String text) {

this.setText(text);

}

/\*\*

\* @return the text

\*/

public String getText() {

return text;

}

/\*\*

\* @param text

\* the text to set

\*/

public void setText(String text) {

this.text = text;

}

}

}

AlertReceiver

package game\_mulot.interfaces;

/\*\*

\* Interface pour les objets susceptibles de recevoir et de traiter des alertes.

\*

\*

\*/

public interface AlertReceiver {

/\*\*

\* Permet de recevoir une alerte et de décider de la traiter ou non.

\*

\* @param alert

\* - l'alert reçue.

\*/

public void receivedAlert(Enum<?> alert);

/\*\*

\* Permet de traiter une alerte.

\*

\* @param alert

\* - l'alerte à traiter.

\*/

public void dealAlert(Enum<?> alert);

}

AlertSender

package game\_mulot.interfaces;

/\*\*

\* Interface pour les objets susceptibles d'envoyer une alertes à un

\* AlertReceiver.

\*

\*

\*/

public interface AlertSender {

/\*\*

\* Envoie une alerte à son parent.

\*

\* @param alert

\* - l'alerte envoyée.

\*/

public void sendAlert(Enum<?> alert);

/\*\*

\* Envoie une alert soit à son parent soit à un autre objet ou les deux.

\*

\* @param alert

\* - l'alerte envoyée.

\* @param receiver

\* - le reeveur de l'alerte, qui n'est pas forcement le parent.

\*/

public void sendAlert(Enum<?> alert, AlertReceiver receiver);

}

Main

package game\_mulot.main;

import java.util.ArrayList;

import java.util.Scanner;

import game\_mulot.basic.BasicComponent;

import game\_mulot.basic.Outils;

import game\_mulot.decors.Socle;

import game\_mulot.niveau.Niveau1;

public class Main {

public static void main(String[] args) {

ArrayList<BasicComponent> list1 = new ArrayList<BasicComponent>();

// list1.add(new Socle(0, 0, 24, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(74, 0, 24, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(3, 3, 15, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(5, 20, 20, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(25, 17, 3, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(25, 16, 5, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(30, 13, 3, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(30, 12, 5, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(34, 13, 3, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(35, 16, 5, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(35, 20, 10, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(45, 18, 2, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(45, 17, 6, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(50, 13, 4, Socle.SocleOrientation.VERTICAL, null));

// list1.add(new Socle(50, 12, 10, Socle.SocleOrientation.HORIZONTAL, null));

// list1.add(new Socle(55, 20, 15, Socle.SocleOrientation.HORIZONTAL, null));

list1.add(new Socle(0, 0, 24, Socle.SocleOrientation.VERTICAL, null));

list1.add(new Socle(74, 0, 24, Socle.SocleOrientation.VERTICAL, null));

list1.add(new Socle(3, 3, 15, Socle.SocleOrientation.HORIZONTAL, null));

list1.add(new Socle(5, 20, 20, Socle.SocleOrientation.HORIZONTAL, null));

list1.add(new Socle(25, 14, 6, Socle.SocleOrientation.VERTICAL, null));

list1.add(new Socle(25, 14, 15, Socle.SocleOrientation.HORIZONTAL, null));

list1.add(new Socle(35, 20, 35, Socle.SocleOrientation.HORIZONTAL, null));

Niveau1 n1 = new Niveau1(list1, null);

@SuppressWarnings("resource")

Scanner sc = new Scanner(System.in);

do {

n1.tic();

n1.draw();

if (Outils.clavier()) {

String s = sc.next();

if (s.length() >= 2){

n1.changeSpecialityMulot(s.charAt(0), s.charAt(1));

}else

System.out.println("il faut une lettre et un chiffre");

}

try {

Thread.sleep(500);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}while (!n1.isFinish());

n1.draw();

n1.tic();

if (n1.win()) {

System.out.println("GAGNE!!!!!");

} else {

System.out.println("PERDU OUUUUUUUUUUUUUUUUUUUUUUUUUUUU!!!");

}

}

}

CelluleEnter

package game\_mulot.niveau;

import java.awt.Color;

import java.awt.Graphics;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

/\*\*

\* Cellule de d'entrée d'un niveau.

\*

\*

\*/

public class CelluleEnter extends BasicCellule {

// Constructors.

public CelluleEnter() {

super();

this.crossable = true;

this.apparence = new Character('E');

}

public CelluleEnter(int posX, int posY, Basic parent) {

super(posX, posY, true, parent);

this.apparence = new Character('E');

}

public CelluleEnter(int posX, int posY, Color color, Object apparence, Basic parent) {

super(posX, posY, true, color, apparence, parent);

}

// Inherited methods.

@Override

public void draw(Graphics g) {

// Non utilisé.

}

@Override

public void draw() {

System.out.print(this.getApparence());

}

@Override

public boolean verifyPosition(int posX, int posY) {

if (this.parent != null) { // Si on a pas un parent null.

// Si notre parent est bien un niveau (RAJOUTER PLUS TARD DES

// CONDITIONS).

if (this.parent.getParent() != null) {

if (this.parent.getParent().getClass() == Niveau.class) {

Niveau n = ((Niveau) this.parent.getParent());

// Si la case où l'on veut aller est une VoidCellule.

if (n.getMatrix()[posY \* n.getColumns() + posY].getClass() == VoidCellule.class) {

return true;

} else {

return false;

}

} else {

return true;

}

} else {

return true;

}

} else {

return true;

}

}

}

CelluleExit

package game\_mulot.niveau;

import java.awt.Color;

import java.awt.Graphics;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

/\*\*

\* Cellule de sortie d'un niveau.

\*

\*

\*/

public class CelluleExit extends BasicCellule {

// Constructors.

public CelluleExit() {

super();

this.crossable = true;

this.apparence = new Character('S');

}

public CelluleExit(int posX, int posY, Basic parent) {

super(posX, posY, true, parent);

this.apparence = new Character('S');

}

public CelluleExit(int posX, int posY, Color color, Object apparence, Basic parent) {

super(posX, posY, true, color, apparence, parent);

}

// Inherited methods.

@Override

public void draw(Graphics g) {

}

@Override

public void draw() {

System.out.print(this.getApparence());

}

@Override

public boolean verifyPosition(int posX, int posY) {

if (this.parent != null) { // Si on a pas un parent null.

// Si notre parent est bien un niveau (RAJOUTER PLUS TARD DES

// CONDITIONS).

if (this.parent.getParent() != null) {

if (this.parent.getParent().getClass() == Niveau.class) {

Niveau n = ((Niveau) this.parent.getParent());

// Si la case où l'on veut aller est une VoidCellule.

if (n.getMatrix()[posY \* n.getColumns() + posY].getClass() == VoidCellule.class) {

return true;

} else {

return false;

}

} else {

return true;

}

} else {

return true;

}

} else {

return true;

}

}

}

Niveau

package game\_mulot.niveau;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicMatrix;

import game\_mulot.basic.BasicComponent;

import game\_mulot.interfaces.AlertReceiver;

import game\_mulot.perso.Mulot;

/\*\*

\* Class qui représente un niveau du jeu.

\*

\*

\*/

public abstract class Niveau extends BasicMatrix implements AlertReceiver {

// Variables.

/\*\*

\* L'entrée du niveau.

\*/

protected NiveauEnter enter;

/\*\*

\* La sortie du niveau.

\*/

protected NiveauExit exit;

/\*\*

\* Le score du joueur.

\*/

protected int score;

/\*\*

\* Nombre de mulots qui le niveau doit créer.

\*/

protected int numberMaxMulot;

/\*\*

\* Nombre minimum de mulots que le joueurs doit faire sortir.

\*/

protected int numberMiniMulot;

/\*\*

\* La liste de composants qui vont être inclus dans la matrice.

\*/

protected ArrayList<BasicComponent> listComponent;

/\*\*

\* <p>

\* Nombre de fois où le niveau a généré un mulot. Il augmente à chaque appel

\* de la fonction <strong>generateMulot</strong>

\* </p>

\*/

protected int numberGenerationMulot;

/\*\*

\* Nombre de mulots morts.

\*/

protected int numberDeadMulot;

/\*\*

\* Nombre de mulots sauvé.

\*/

protected int numberSaveMulot;

/\*\*

\* La liste des Mulots.

\*/

protected ArrayList<Mulot> listMulot;

protected Enum<?> currentAlert;

// Constructors.

public Niveau() {

super();

this.enter = new NiveauEnter(0, 0, this);

this.exit = new NiveauExit(this.getColumns() - 1, this.getLines() - 1, this);

this.score = 0;

this.numberMaxMulot = 1;

this.numberMiniMulot = 1;

this.numberGenerationMulot = 0;

this.numberDeadMulot = 0;

this.numberSaveMulot = 0;

this.currentAlert = null;

this.listMulot = new ArrayList<Mulot>();

this.listComponent = new ArrayList<BasicComponent>();

this.listComponent.add(this.enter);

this.listComponent.add(this.exit);

this.includeComponentInMatrix();

}

@SuppressWarnings("unchecked")

public Niveau(String name, int lines, int columns, ArrayList<BasicComponent> listComponent, int nombreMaxMulot,

int nombreMiniMulot, Basic parent) {

super(name, lines, columns, listComponent, parent);

this.enter = new NiveauEnter(0, 0, this);

this.exit = new NiveauExit(this.getColumns() - 1, this.getLines() - 1, this);

this.score = 0;

this.numberMaxMulot = nombreMaxMulot;

this.numberMiniMulot = nombreMiniMulot;

this.numberGenerationMulot = 0;

this.numberDeadMulot = 0;

this.numberSaveMulot = 0;

this.currentAlert = null;

this.listMulot = new ArrayList<Mulot>();

this.listComponent = new ArrayList<BasicComponent>();

// On donne à tous les composants comme parent le niveau.

if (listComponent != null) {

this.listComponent = (ArrayList<BasicComponent>) listComponent.clone();

for (BasicComponent component : this.getListComponent()) {

component.setParent(this);

}

}

this.listComponent.add(this.enter);

this.listComponent.add(this.exit);

this.includeComponentInMatrix();

}

// Abstract methods.

/\*\*

\*

\* @return true si le niveau est fini sinon false;

\*/

public abstract boolean isFinish();

/\*\*

\* Fait tourner le jeu.

\*/

public abstract void tic();

/\*\*

\* Génère un Mulot.

\*/

protected abstract void generateMulot();

/\*\*

\* Change la matrice avec les nouveaux changements.

\*/

public abstract void refresh();

/\*\*

\* Permet de vérifier si le niveau est gagner ou non.

\*

\* @return true si le niveau est gagner sinon false.

\*/

public abstract boolean win();

/\*\*

\* Change le mulot qui a le numéro entré en paramètre en la spécialité entré

\* en paramètre.

\*

\* @param numberMulot

\* - le numeros en caractère du mulot.

\* @param speciality

\* - la spécialité en caratère que le mulot doit prendre.

\*/

public abstract void changeSpecialityMulot(char numberMulot, char speciality);

/\*\*

\* Permet de chercher un mulot qui porte le numéro entré en paramètre.

\*

\* @param numberMulot

\* - le numéro en caractère du mulot.

\*

\* @return le mulot qui porte le numéro entré en paramètre.

\*/

protected abstract Mulot mulotAt(char numberMulot);

// Inherited methods.

@Override

public void receivedAlert(Enum<?> alert) {

if (this.getListMulot().contains((Mulot) ((Mulot.MulotAlert) alert).getSender())) {

this.setCurrentAlert((Mulot.MulotAlert) alert);

}

}

@Override

public void dealAlert(Enum<?> alert) {

Mulot.MulotAlert mulotAlert = (Mulot.MulotAlert) alert;

if (mulotAlert == Mulot.MulotAlert.CHANGE\_SPECIALITY) {

// On change le mulot de spécialité.

this.listMulot.set(this.listMulot.indexOf(mulotAlert.getSender()), mulotAlert.getNewSpeciality());

}

// On remet l'alert à null.

this.setCurrentAlert(null);

}

@Override

protected void initializeMatrix() {

this.matrix = new BasicCellule[this.getLines() \* this.getColumns()];

for (int i = 0; i < this.getLines(); i++) { // Parcours des lignes.

for (int j = 0; j < this.getColumns(); j++) { // Parcours des

// colonnes.

this.matrix[i \* this.getColumns() + j] = new VoidCellule(j, i, this);

}

}

}

@Override

protected void includeComponentInMatrix() {

// On met d'abord les composants.

for (BasicComponent component : this.getListComponent()) {

for (BasicCellule componentCellule : component.getListModelisation()) {

// On remplace la cellule vide par la cellule du composant.

this.matrix[componentCellule.getPosY() \* this.getColumns()

+ componentCellule.getPosX()] = componentCellule;

}

}

// On met ensuite les mulots.

for (BasicComponent component : this.getListMulot()) {

for (BasicCellule componentCellule : component.getListModelisation()) {

// On remplace la cellule vide par la cellule du composant.

this.matrix[componentCellule.getPosY() \* this.getColumns()

+ componentCellule.getPosX()] = componentCellule;

}

}

}

// Public methods.

@Override

public String toString() {

return "Niveau [" + "score=" + score + ", nombreMaxMulot=" + numberMaxMulot + ", nombreMiniMulot="

+ numberMiniMulot + ", numberGenerationMulot=" + numberGenerationMulot + ", numberDeadMulot="

+ numberDeadMulot + ", numberSaveMulot=" + numberSaveMulot + ", listMulot=" + listMulot + "]";

}

/\*\*

\* Ajouter un composant dans la liste des composants de la matrice.

\*

\* @param component

\* - le composant à aujouter.

\*/

public void addComponent(BasicComponent component) {

component.setParent(this);

this.listComponent.add(component);

}

/\*\*

\* Supprime un composant dans la liste des composants de la matrice.

\*

\* @param component

\* - le composant à supprimer.

\*/

public void removeComponent(BasicComponent component) {

if (component.getParent() == this)

component.setParent(null);

this.listComponent.remove(component);

}

// Getters and Setters.

/\*\*

\* @return the enter

\*/

public NiveauEnter getEnter() {

return enter;

}

/\*\*

\* @param enter

\* the enter to set

\*/

public void setEnter(NiveauEnter enter) {

this.enter = enter;

if (this.enter != null)

this.enter.setParent(this);

else

this.enter = new NiveauEnter(this.getColumns() - 1, this.getLines() - 1, this);

}

/\*\*

\* @return the exit

\*/

public NiveauExit getExit() {

return exit;

}

/\*\*

\* @param exit

\* the exit to set

\*/

public void setExit(NiveauExit exit) {

this.exit = exit;

if (this.exit != null)

this.exit.setParent(this);

else

this.exit = new NiveauExit(this.getColumns() - 1, this.getLines() - 1, this);

}

/\*\*

\* @return the score

\*/

public int getScore() {

return score;

}

/\*\*

\* @param score

\* the score to set

\*/

public void setScore(int score) {

this.score = score;

}

/\*\*

\* @return the nombreMaxMulot

\*/

public int getNumberMaxMulot() {

return numberMaxMulot;

}

/\*\*

\* @return the nombreMiniMulot

\*/

public int getNumberMiniMulot() {

return numberMiniMulot;

}

/\*\*

\* @return the listMulot

\*/

public ArrayList<Mulot> getListMulot() {

return listMulot;

}

/\*\*

\* @return the numberGenerationMulot

\*/

public int getNumberGenerationMulot() {

return numberGenerationMulot;

}

/\*\*

\* @return the numberDeadMulot

\*/

public int getNumberDeadMulot() {

return numberDeadMulot;

}

/\*\*

\* @param numberDeadMulot

\* the numberDeadMulot to set

\*/

public void setNumberDeadMulot(int numberDeadMulot) {

this.numberDeadMulot = numberDeadMulot;

}

/\*\*

\* @return the numberSaveMulot

\*/

public int getNumberSaveMulot() {

return numberSaveMulot;

}

/\*\*

\* @param numberSaveMulot

\* the numberSaveMulot to set

\*/

public void setNumberSaveMulot(int numberSaveMulot) {

this.numberSaveMulot = numberSaveMulot;

}

/\*\*

\* @return the currentAlert

\*/

public Enum<?> getCurrentAlert() {

return currentAlert;

}

/\*\*

\* @param currentAlert

\* the currentAlert to set

\*/

public void setCurrentAlert(Enum<?> currentAlert) {

this.currentAlert = currentAlert;

}

/\*\*

\* @return the listComponent

\*/

public ArrayList<BasicComponent> getListComponent() {

return listComponent;

}

}

Niveau1

package game\_mulot.niveau;

import java.awt.Graphics;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicComponent;

import game\_mulot.perso.Mulot;

import game\_mulot.perso.MulotGrimpeur;

import game\_mulot.perso.MulotParachute;

/\*\*

\* Le niveau 1.

\*

\*

\*/

public class Niveau1 extends Niveau {

protected int numberMaxGrimpeur = 10;

protected int numberMaxParachute = 10;

// Constructors.

public Niveau1() {

super("Niveau 1", 10, 10, null, 1, 1, null);

this.listComponent.remove(this.enter);

this.listComponent.remove(this.exit);

this.enter = new NiveauEnter(3, 0, this);

this.exit = new NiveauExit(9, 8, this);

this.listComponent.add(this.enter);

this.listComponent.add(this.exit);

this.refresh();

}

public Niveau1(ArrayList<BasicComponent> listComponent, Basic parent) {

super("Niveau 1", 25, 75, listComponent, 10, 1, parent);

this.listComponent.remove(this.enter);

this.listComponent.remove(this.exit);

this.enter = new NiveauEnter(10, 0, this);

this.exit = new NiveauExit(69, 19, this);

this.listComponent.add(this.enter);

this.listComponent.add(this.exit);

this.refresh();

}

// Inhereted methods.

@Override

public boolean isFinish() {

// On vérifie si le nombre de mulots morts n'empêche pas de gagner.

if (this.getNumberMaxMulot() - this.getNumberDeadMulot() >= this.getNumberMiniMulot()) {

// Si tous les mulots sont sortie.

if (this.getNumberSaveMulot() == this.getNumberMaxMulot()) {

return true;

} else if (this.getScore() >= this.getNumberMiniMulot() && this.getListMulot().isEmpty()) {

return true;

} else {

return false;

}

} else {

return true;

}

}

@Override

public void tic() {

// On crée un nouveau mulot.

this.generateMulot();

// On fait avancé tout les mulots créés.

for (Mulot mulot : this.getListMulot()) {

mulot.advance(1);

}

// On vérifie si il n'y a pas eu d'alert.

if (this.getCurrentAlert() != null) {

this.dealAlert(this.getCurrentAlert());

}

// On rafraîchi le niveau.

this.refresh();

}

@Override

protected void generateMulot() {

// On vérifie si on à pas généré le nombre maximum de mulot.

if (this.getNumberGenerationMulot() < this.getNumberMaxMulot()) {

Mulot m = new Mulot(String.valueOf(this.getNumberGenerationMulot()), this.getEnter().getPosX(),

this.getEnter().getPosY(), this);

for (BasicCellule cellule : m.getListModelisation()) {

cellule.setApparence(m.getName().charAt(0));

}

this.listMulot.add(m);

this.numberGenerationMulot++;

}

}

@Override

public void refresh() {

// On réinitialise la matrice.

this.initializeMatrix();

// On nettoie la liste de Mulot (tous les morts sont remove.

for (int i = 0; i < this.getListMulot().size(); i++) {

if (this.getListMulot().get(i).getStatue() == Mulot.MulotStatue.DEAD) {

this.getListMulot().remove(this.getListMulot().get(i));

}

}

// On met les composants avec leur nouvelle place.

this.includeComponentInMatrix();

}

@Override

public boolean win() {

if (this.getScore() >= this.getNumberMiniMulot()) {

return true;

} else {

return false;

}

}

/\*\*

\* Cherche dans la liste des mulot un mulot ayant pour numéros le numéros

\* indiqué. Si le mulot n'est pas trouvé dans la liste, la méthod return

\* null.

\*/

@Override

protected Mulot mulotAt(char numberMulot) {

Mulot m = null;

// On cherche le mulot dans la liste de mulot.

for (int i = 0; i < this.getListMulot().size(); i++) {

boolean escape = false;

// Si le mulot a bien le numéro que l'on cherche.

if (this.getListMulot().get(i).getName().charAt(0) == numberMulot) {

m = this.getListMulot().get(i);

escape = true;

}

if (escape)

break;

}

return m;

}

@Override

public void changeSpecialityMulot(char numberMulot, char speciality) {

switch (speciality) {

case 'g':

Mulot m1 = this.mulotAt(numberMulot);

// On change le mulot.

if (this.numberMaxGrimpeur > 0 && m1 != null) {

MulotGrimpeur mg = new MulotGrimpeur(m1.getName(), m1.getPosX(), m1.getPosY(), m1.getDirection(), m1.getParent());

for (BasicCellule cellule : mg.getListModelisation()) {

cellule.setApparence(m1.getName().charAt(0));

}

m1.changeSpeciality(mg);

this.numberMaxGrimpeur--;

}

break;

case 'p':

Mulot m2 = this.mulotAt(numberMulot);

// On change le mulot.

if (this.numberMaxParachute > 0 && m2 != null) {

MulotParachute mp = new MulotParachute(m2.getName(), m2.getPosX(), m2.getPosY(), m2.getDirection(), m2.getParent());

for (BasicCellule cellule : mp.getListModelisation()) {

cellule.setApparence(m2.getName().charAt(0));

}

m2.changeSpeciality(mp);

this.numberMaxParachute--;

}

break;

default:

System.out.println("Tu t'es trompé de lettre");

break;

}

}

@Override

public void draw(Graphics g) {

}

@Override

public void draw() {

this.includeComponentInMatrix();

for (int i = 0; i < this.getLines(); i++) {

for (int j = 0; j < this.getColumns(); j++) {

this.matrix[i \* this.getColumns() + j].draw();

}

System.out.println();

}

System.out.println("G : " + this.numberMaxGrimpeur + "\tP : " + this.numberMaxParachute);

}

// Getters and Setters.

/\*\*

\* @return the numberMaxGrimpeur

\*/

public int getNumberMaxGrimpeur() {

return numberMaxGrimpeur;

}

/\*\*

\* @param numberMaxGrimpeur

\* the numberMaxGrimpeur to set

\*/

public void setNumberMaxGrimpeur(int numberMaxGrimpeur) {

this.numberMaxGrimpeur = numberMaxGrimpeur;

}

/\*\*

\* @return the numberMaxParachute

\*/

public int getNumberMaxParachute() {

return numberMaxParachute;

}

/\*\*

\* @param numberMaxParachute

\* the numberMaxParachute to set

\*/

public void setNumberMaxParachute(int numberMaxParachute) {

this.numberMaxParachute = numberMaxParachute;

}

}

NiveauEntrer

package game\_mulot.niveau;

import java.awt.Graphics;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicComponent;

/\*\*

\* Entrée d'un niveau.

\*

\*/

public class NiveauEnter extends BasicComponent {

public NiveauEnter() {

super();

}

public NiveauEnter(int posX, int posY, Basic Parent) {

super("Enter", posX, posY, Parent);

}

@Override

protected void initializeListModelisation() {

this.listModelisation = new ArrayList<BasicCellule>();

this.listModelisation.add(new CelluleEnter(this.getPosX(), this.getPosY(), this));

}

@Override

public void draw(Graphics g) {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw(g);

}

}

@Override

public void draw() {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw();

}

}

@Override

public boolean verifyPosition(int posX, int posY) {

boolean verif = false;

if (this.getListModelisation() != null) {

for (BasicCellule cellule : this.getListModelisation()) {

verif = cellule.verifyPosition(posX - (this.getPosX() - cellule.getPosX()),

posY - (this.getPosY() - cellule.getPosY()));

if (!verif) {

break;

}

}

} else {

verif = true;

}

return verif;

}

}

NiveauExit

package game\_mulot.niveau;

import java.awt.Graphics;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicComponent;

/\*\*

\* Sortie d'un niveau.

\*

\*

\*/

public class NiveauExit extends BasicComponent {

public NiveauExit() {

super();

}

public NiveauExit(int posX, int posY, Basic Parent) {

super("Exit", posX, posY, Parent);

}

@Override

protected void initializeListModelisation() {

this.listModelisation = new ArrayList<BasicCellule>();

this.listModelisation.add(new CelluleExit(this.getPosX(), this.getPosY(), this));

}

@Override

public void draw(Graphics g) {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw(g);

}

}

@Override

public void draw() {

for (BasicCellule cellule : this.listModelisation) {

cellule.draw();

}

}

@Override

public boolean verifyPosition(int posX, int posY) {

boolean verif = false;

if (this.getListModelisation() != null) {

for (BasicCellule cellule : this.getListModelisation()) {

verif = cellule.verifyPosition(posX - (this.getPosX() - cellule.getPosX()),

posY - (this.getPosY() - cellule.getPosY()));

if (!verif) {

break;

}

}

} else {

verif = true;

}

return verif;

}

}

VoidCellule

package game\_mulot.niveau;

import java.awt.Color;

import java.awt.Graphics;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

/\*\*

\* Cellule qui représente le vide donc toujours traversable. Par défaut

\* l'apparence est un Character(' ').

\*

\*

\*/

public class VoidCellule extends BasicCellule {

public VoidCellule() {

super();

this.crossable = true;

this.apparence = new Character(' ');

}

public VoidCellule(int posX, int posY, Basic parent) {

super(posX, posY, true, parent);

this.apparence = new Character(' ');

}

public VoidCellule(int posX, int posY, Color color, Object apparence, Basic parent) {

super(posX, posY, true, color, apparence, parent);

}

@Override

public void draw(Graphics g) {

// Non utilisé.

}

@Override

public void draw() {

System.out.print(this.getApparence());

}

@Override

public boolean verifyPosition(int posX, int posY) {

return true;

}

}

Mulot

package game\_mulot.perso;

import java.awt.Graphics;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.interfaces.AlertReceiver;

import game\_mulot.interfaces.AlertSender;

import game\_mulot.niveau.Niveau;

import game\_mulot.niveau.VoidCellule;

/\*\*

\* <p>

\* La class qui représente le mulot.

\* </p>

\* <p>

\* <strong>Précision :</strong>Il faut prendre en compte qu'ici les Y

\* représentent les lignes et que les X représentent les colonnes

\* </p>

\*

\*

\*

\*/

public class Mulot extends Personage implements AlertReceiver, AlertSender {

// Variables.

/\*\*

\* La direction dans laquelle se dirige le mulot.

\*/

protected Mulot.MulotDirection direction;

/\*\*

\* Le statue du mulot, mort ou vivant.

\*/

protected Mulot.MulotStatue statue;

/\*\*

\* Alerte courante toujours null sauf quand il y a une alert.

\*/

protected MulotCellule.MulotCelluleAlert currentAlert;

/\*\*

\* Compteur pour compter le nombre de cases de chute.

\*/

protected int fallCounter;

// Constructors.

public Mulot() {

super();

this.name = "Mulot";

this.direction = Mulot.MulotDirection.SOUTH; // On tombe de l'entrée.

this.statue = Mulot.MulotStatue.ALIVE;

this.setCurrentAlert(null);

this.fallCounter = 0;

}

public Mulot(String name, int posX, int posY, Basic Parent) {

super(name, posX, posY, Parent);

this.direction = Mulot.MulotDirection.SOUTH; // On tombe de l'entrée.

this.statue = Mulot.MulotStatue.ALIVE;

this.setCurrentAlert(null);

this.fallCounter = 0;

}

public Mulot(String name, int posX, int posY, Mulot.MulotDirection direction, Basic Parent) {

super(name, posX, posY, Parent);

this.direction = direction;

if (this.direction == null) {

this.direction = Mulot.MulotDirection.SOUTH;

}

this.statue = Mulot.MulotStatue.ALIVE;

this.setCurrentAlert(null);

this.fallCounter = 0;

}

// Inherited methods.

@Override

public void receivedAlert(Enum<?> alert) {

if (this.getListModelisation().contains((MulotCellule) ((MulotCellule.MulotCelluleAlert) alert).getSender())) {

this.setCurrentAlert((MulotCellule.MulotCelluleAlert) alert);

}

}

@Override

public void dealAlert(Enum<?> alert) {

MulotCellule.MulotCelluleAlert alertMulot = ((MulotCellule.MulotCelluleAlert) alert);

if (alertMulot == MulotCellule.MulotCelluleAlert.COLLISION) {

if (this.getDirection() == Mulot.MulotDirection.NORTH) {

this.setDirection(Mulot.MulotDirection.SOUTH);

} else if (this.getDirection() == Mulot.MulotDirection.SOUTH) {

if (this.getFallCounter() >= 10) {

this.setStatue(Mulot.MulotStatue.DEAD);

} else {

this.setDirection(Mulot.MulotDirection.EAST);

this.fallCounter = 0;

}

} else if (this.getDirection() == Mulot.MulotDirection.EAST) {

this.setDirection(Mulot.MulotDirection.WEST);

} else if (this.getDirection() == Mulot.MulotDirection.WEST) {

this.setDirection(Mulot.MulotDirection.EAST);

}

} else if (alertMulot == MulotCellule.MulotCelluleAlert.ARRIVED) {

((Niveau) this.getParent()).setNumberSaveMulot(((Niveau) this.getParent()).getNumberSaveMulot() + 1);

((Niveau) this.getParent()).setScore(((Niveau) this.getParent()).getScore() + 1);

// On le ren dmort pour qu'il n'avance plus.

this.statue = Mulot.MulotStatue.DEAD;

} else if (alertMulot == MulotCellule.MulotCelluleAlert.OUT\_OF\_MAP) {

this.setStatue(Mulot.MulotStatue.DEAD);

}

// On remet l'alerte courante à null.

this.currentAlert = null;

}

@Override

public void sendAlert(Enum<?> alert) {

((Mulot.MulotAlert) alert).setSender(this);

((AlertReceiver) this.getParent()).receivedAlert(alert);

}

@Override

public void sendAlert(Enum<?> alert, AlertReceiver receiver) {

((Mulot.MulotAlert) alert).setSender(this);

receiver.receivedAlert(alert);

}

/\*\*

\* Déplace le mulot vers le haut.

\*/

@Override

public void moveUp(int deplacement) {

this.setPosY(this.getPosY() - deplacement);

// Si il y a eu une alerte, l'objet n'a pas bouger et on traite alors

// l'alerte.

if (this.getCurrentAlert() != null) {

// On traite l'alerte.

this.dealAlert(this.getCurrentAlert());

}

}

/\*\*

\* Déplace le mulot vers le bas.

\*/

@Override

public void moveDown(int deplacement) {

this.setPosY(this.getPosY() + deplacement);

// Si il y a eu une alerte, l'objet n'a pas bouger et on traite alors

// l'alerte.

if (this.getCurrentAlert() != null) {

// On traite l'alerte.

this.dealAlert(this.getCurrentAlert());

} else {

// On ajoute le deplacement au counter de chute car on tombe.

this.fallCounter += deplacement;

}

}

/\*\*

\* Déplace le mulot vers la droite.

\*/

@Override

public void moveRight(int deplacement) {

this.setPosX(this.getPosX() + deplacement);

// Si il y a eu une alerte, l'objet n'a pas bouger et on traite alors

// l'alerte.

if (this.getCurrentAlert() != null) {

// On traite l'alerte.

this.dealAlert(this.getCurrentAlert());

}

}

/\*\*

\* Déplace le mulot vers la gauche.

\*/

@Override

public void moveLeft(int deplacement) {

this.setPosX(this.getPosX() - deplacement);

// Si il y a eu une alerte, l'objet n'a pas bouger et on traite alors

// l'alerte.

if (this.getCurrentAlert() != null) {

// On traite l'alerte.

this.dealAlert(this.getCurrentAlert());

}

}

@Override

protected void initializeListModelisation() {

this.listModelisation = new ArrayList<BasicCellule>();

this.listModelisation.add(new MulotCellule(this.getPosX(), this.getPosY(), this));

}

@Override

public void draw(Graphics g) {

if (this.getListModelisation() != null) {

for (BasicCellule cellule : this.getListModelisation()) {

cellule.draw(g);

}

}

}

@Override

public void draw() {

if (this.getListModelisation() != null) {

for (BasicCellule cellule : this.getListModelisation()) {

cellule.draw();

}

}

}

@Override

public boolean verifyPosition(int posX, int posY) {

boolean verif = false;

if (this.getListModelisation() != null) {

for (BasicCellule cellule : this.getListModelisation()) {

verif = cellule.verifyPosition(posX - (this.getPosX() - cellule.getPosX()),

posY - (this.getPosY() - cellule.getPosY()));

if (!verif) {

break;

}

}

} else {

verif = true;

}

return verif;

}

// Public methods.

/\*\*

\* Déplace le mulot d'autant de case que de déplacement entré en paramètre

\* dans la direction qu'il possède actuellement.

\*

\* @param deplacement

\* - le nombre de déplacement du mulot.

\*/

public void advance(int deplacement) {

// On vérifie l'état du mulot.

if (this.getStatue() != Mulot.MulotStatue.DEAD) {

// On vérifie si il y a un sol en dessous du mulot.

try {

BasicCellule celluleDessous = ((Niveau) this.getParent())

.getMatrix()[(this.getListModelisation().get(0).getPosY() + 1)

\* ((Niveau) this.getParent()).getColumns()

+ (this.getListModelisation().get(0).getPosX())];

if (celluleDessous.getClass() == VoidCellule.class || celluleDessous.getClass() == MulotCellule.class) {

this.setDirection(Mulot.MulotDirection.SOUTH);

}

}

// Si on cherche endehors de la map

catch (ArrayIndexOutOfBoundsException e) {

}

// On fait bouger dans tous les cas le mulot.

finally {

if (this.getDirection() == Mulot.MulotDirection.NORTH) {

this.moveUp(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.SOUTH) {

this.moveDown(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.EAST) {

this.moveRight(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.WEST) {

this.moveLeft(deplacement);

}

}

}

}

/\*\*

\* Change la spécialité du mulot. Envoie un ealerte au niveau dans lequel on

\* est pour qu'il puisse changer la spécialité du mulot.

\*

\* @param newSpecility

\* - la nouvelle spécialité du mulot.

\*/

public void changeSpeciality(Mulot newSpeciality) {

// On vérifie si le mulot est bien dans un niveau

if (this.getParent() != null) {

Mulot.MulotAlert alert = Mulot.MulotAlert.CHANGE\_SPECIALITY;

alert.setSender(this);

alert.setNewSpeciality(newSpeciality);

((Niveau) this.getParent()).receivedAlert(alert);

}

}

// Getters and Setters.

/\*\*

\* @return the direction

\*/

public Mulot.MulotDirection getDirection() {

return direction;

}

/\*\*

\* @param direction

\* the direction to set

\*/

public void setDirection(Mulot.MulotDirection direction) {

this.direction = direction;

}

/\*\*

\* @return the statue

\*/

public Mulot.MulotStatue getStatue() {

return statue;

}

/\*\*

\* @param statue

\* the statue to set

\*/

public void setStatue(Mulot.MulotStatue statue) {

this.statue = statue;

// A CHANGER EN FAISANT IMPLEMENTER LES OBJECTS BASIC AVEC LES INTERFACE

// SERNDER ET RECEIVER ALERT!!!

if (this.statue == Mulot.MulotStatue.DEAD) {

// On le supprime de la liste.

((Niveau) this.parent).setNumberDeadMulot(((Niveau) this.parent).getNumberDeadMulot() + 1);

}

}

/\*\*

\* @return the currentAlert

\*/

public MulotCellule.MulotCelluleAlert getCurrentAlert() {

return currentAlert;

}

/\*\*

\* @param currentAlert

\* the currentAlert to set

\*/

public void setCurrentAlert(MulotCellule.MulotCelluleAlert currentAlert) {

this.currentAlert = currentAlert;

}

/\*\*

\* @return the fallCounter

\*/

public int getFallCounter() {

return fallCounter;

}

// Enumerations.

/\*\*

\* <p>

\* Enumeration qui contient toute les direction dans laquelle un Mulot peut

\* se diriger.

\* </p>

\*

\* <p>

\* <strong>Liste des directions :</br>

\* </strong>

\* <ul>

\* <li>NORTH</li>

\* <li>SOUTH</li>

\* <li>EAST</li>

\* <li>WEST</li>

\* </ul>

\* </p>

\*

\*

\* @author Calli

\*

\*/

public enum MulotDirection {

NORTH("NORTH"), // La direction NORD (haut).

SOUTH("SOUTH"), // La direction SUD (bas).

EAST("EAST"), // La directtion EST (droite).

WEST("WEST"); // La direction OUEST (gauche).

private String text = "";

MulotDirection(String text) {

this.setText(text);

}

/\*\*

\* @return the text

\*/

public String getText() {

return text;

}

/\*\*

\* @param text

\* the text to set

\*/

public void setText(String text) {

this.text = text;

}

}

public enum MulotStatue {

DEAD("DEAD"), // Statue mort.

ALIVE("ALIVE"); // Statue vivant.

private String text = "";

MulotStatue(String text) {

this.setText(text);

}

/\*\*

\* @return the text

\*/

public String getText() {

return text;

}

/\*\*

\* @param text

\* the text to set

\*/

public void setText(String text) {

this.text = text;

}

}

public enum MulotAlert {

CHANGE\_SPECIALITY("CHANGE\_SPECIALITY");

private String text = "";

private Object sender = null;

private Mulot newSpeciality = null;

MulotAlert(String text) {

this.setText(text);

this.sender = null;

this.newSpeciality = null;

}

/\*\*

\* @return the text

\*/

public String getText() {

return text;

}

/\*\*

\* @param text

\* the text to set

\*/

public void setText(String text) {

this.text = text;

}

/\*\*

\* @return the sender

\*/

public Object getSender() {

return sender;

}

/\*\*

\* @param sender

\* the sender to set

\*/

public void setSender(Object sender) {

this.sender = sender;

}

/\*\*

\* @return the newSpeciality

\*/

public Mulot getNewSpeciality() {

return newSpeciality;

}

/\*\*

\* @param newSpeciality

\* the newSpeciality to set

\*/

public void setNewSpeciality(Mulot newSpeciality) {

this.newSpeciality = newSpeciality;

}

}

}

MulotCellule

package game\_mulot.perso;

import java.awt.Color;

import java.awt.Graphics;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.basic.BasicMatrix;

import game\_mulot.interfaces.AlertReceiver;

import game\_mulot.interfaces.AlertSender;

import game\_mulot.niveau.CelluleExit;

import game\_mulot.niveau.Niveau;

import game\_mulot.niveau.VoidCellule;

/\*\*

\* Clas qui représente une cellule de mulot.

\*

\*

\*/

public class MulotCellule extends BasicCellule implements AlertSender {

/\*\*

\* On initialise avec le constructor de la class mère et on met l'apparence

\* à un Character('M'). La cellule les n'est pas traversable.

\*/

public MulotCellule() {

super();

this.setApparence(new Character('M'));

}

/\*\*

\* Initialise avec un parent, une position X et une position Y et donne pour

\* apparence un Character('M'). La cellule n'est pas traversable.

\*

\* @param posX

\* - la position X de la cellule.

\* @param posY

\* - la position Y de la cellule.

\* @param parent

\* - le parent de la cellule.

\*/

public MulotCellule(int posX, int posY, Basic parent) {

super(posX, posY, false, parent);

this.setCrossable(false);

this.setParent(parent);

this.setApparence(new Character('M'));

}

public MulotCellule(int posX, int posY, Color color, Object apparence, Basic parent) {

super(posX, posY, false, color, apparence, parent);

}

// Abstract methods.

@Override

public void sendAlert(Enum<?> alert) {

((MulotCellule.MulotCelluleAlert) alert).setSender(this);

((AlertReceiver) this.getParent()).receivedAlert(alert);

}

@Override

public void sendAlert(Enum<?> alert, AlertReceiver receiver) {

((MulotCellule.MulotCelluleAlert) alert).setSender(this);

receiver.receivedAlert(alert);

}

@Override

public void draw(Graphics g) {

// Non utiliser.

}

@Override

public void draw() {

System.out.print(this.getApparence());

}

@Override

public boolean verifyPosition(int posX, int posY) {

// On vérifie si notre parent n'est pas null.

if (this.getParent() != null) {

// On vérifie ensuite si on est bien dans une matrice.

if (this.getParent().getParent() != null) {

BasicMatrix matrix = (BasicMatrix) this.getParent().getParent();

// On vérifie alors si on est bien dans la map.

if ((posX >= 0 && posX < matrix.getColumns())

&& (posY >= 0 && posY < matrix.getLines())) {

Niveau niveau = (Niveau) matrix;

// La cellule où l'on veut aller.

BasicCellule cellule = niveau.getMatrix()[posY \* niveau.getColumns() + posX];

if (cellule.getClass() == CelluleExit.class) {

this.sendAlert(MulotCellule.MulotCelluleAlert.ARRIVED);

return true;

// On vérifie si la cellule n'est pas une cellule vide

} else if (cellule.getClass() == VoidCellule.class) {

return true;

// On vérifie si il y n'y pas collision.

} else if (cellule.isCrossable()) {

return true;

// On vérifie si la cellule n'est pas une cellule d'un

// autre mulot. Les mulots peuvent être sur la meme

// case.

} else if (cellule.getClass() == MulotCellule.class) {

return true;

// Sinon on ne peut pas bouger, il y a une collision.

} else {

this.sendAlert(MulotCellule.MulotCelluleAlert.COLLISION);

return false;

}

} else {

this.sendAlert(MulotCellule.MulotCelluleAlert.OUT\_OF\_MAP);

return false;

}

} else {

// Ici c'est lorsque l'on créer le Mulot ou la map.

return true;

}

} else {

// ici c'est une cellule quelconque sans parent sans rien donc comme

// si c'etait le constructeur pas défault.

return true;

}

}

// Enumerations.

public enum MulotCelluleAlert {

COLLISION("COLLISION"), // Alerte lorsque la cellule entre en collision.

OUT\_OF\_MAP("OUT\_OF\_MAP"), // Alerte lorsque le cellule est hors de la

// map.

ARRIVED("ARRIVED"); // Alerte lorsque la cellule est sur la case

// arrivée.

private String text = "";

private Object sender;

MulotCelluleAlert(String text) {

this.setText(text);

this.setSender(null);

}

/\*\*

\* @return the text

\*/

public String getText() {

return text;

}

/\*\*

\* @param text

\* the text to set

\*/

public void setText(String text) {

this.text = text;

}

/\*\*

\* @return the sender

\*/

public Object getSender() {

return sender;

}

/\*\*

\* @param sender

\* the sender to set

\*/

public void setSender(Object sender) {

this.sender = sender;

}

}

}

MulotGrimpeur

package game\_mulot.perso;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.niveau.Niveau;

import game\_mulot.niveau.VoidCellule;

/\*\*

\* <p>

\* Le mulot grimpeur qui sait grimper aux murs.

\* </p>

\*

\*

\*/

public class MulotGrimpeur extends Mulot {

// Constructors.

public MulotGrimpeur() {

super();

}

public MulotGrimpeur(String name, int posX, int posY, Basic Parent) {

super(name, posX, posY, Parent);

}

public MulotGrimpeur(String name, int posX, int posY, Mulot.MulotDirection direction, Basic parent) {

super(name, posX, posY, direction, parent);

}

// Override methods

@Override

protected void initializeListModelisation() {

this.listModelisation = new ArrayList<BasicCellule>();

this.listModelisation.add(new MulotCellule(this.getPosX(), this.getPosY(), null, new Character('G'), this));

}

@Override

public void dealAlert(Enum<?> alert) {

MulotCellule.MulotCelluleAlert alertMulot = ((MulotCellule.MulotCelluleAlert) alert);

if (alertMulot == MulotCellule.MulotCelluleAlert.COLLISION) {

if (this.getDirection() == Mulot.MulotDirection.NORTH) {

this.setDirection(Mulot.MulotDirection.SOUTH);

} else if (this.getDirection() == Mulot.MulotDirection.SOUTH) {

if (this.getFallCounter() >= 10) {

this.setStatue(Mulot.MulotStatue.DEAD);

} else {

this.setDirection(Mulot.MulotDirection.EAST);

this.fallCounter = 0;

}

} else if (this.getDirection() == Mulot.MulotDirection.EAST) {

this.setDirection(Mulot.MulotDirection.NORTH);

} else if (this.getDirection() == Mulot.MulotDirection.WEST) {

this.setDirection(Mulot.MulotDirection.EAST);

}

} else if (alertMulot == MulotCellule.MulotCelluleAlert.ARRIVED) {

((Niveau) this.getParent()).setNumberSaveMulot(((Niveau) this.getParent()).getNumberSaveMulot() + 1);

((Niveau) this.getParent()).setScore(((Niveau) this.getParent()).getScore() + 1);

// On le ren dmort pour qu'il n'avance plus.

this.statue = Mulot.MulotStatue.DEAD;

} else if (alertMulot == MulotCellule.MulotCelluleAlert.OUT\_OF\_MAP) {

this.setStatue(Mulot.MulotStatue.DEAD);

}

// On remet l'alerte courante à null.

this.currentAlert = null;

}

@Override

public void advance(int deplacement) {

// On vérifie l'état du mulot.

if (this.getStatue() != Mulot.MulotStatue.DEAD) {

try {

// On vérifie si il y a un sol en dessous du mulot et que le

// mulot

// ne soit pas en train de grimper.

if (this.getDirection() != Mulot.MulotDirection.NORTH) {

BasicCellule celluleDessous = ((Niveau) this.getParent())

.getMatrix()[(this.getListModelisation().get(0).getPosY() + 1)

\* ((Niveau) this.getParent()).getColumns()

+ (this.getListModelisation().get(0).getPosX())];

if (celluleDessous.getClass() == VoidCellule.class

|| celluleDessous.getClass() == MulotCellule.class) {

this.setDirection(Mulot.MulotDirection.SOUTH);

}

}

// On vérifie si le mulot n'est pas arriver au bout du mur.

else {

BasicCellule celluleDroite = ((Niveau) this.getParent())

.getMatrix()[this.getListModelisation().get(0).getPosY()

\* ((Niveau) this.getParent()).getColumns()

+ (this.getListModelisation().get(0).getPosX() + 1)];

if (celluleDroite.getClass() == VoidCellule.class

|| celluleDroite.getClass() == MulotCellule.class) {

this.setDirection(Mulot.MulotDirection.EAST);

// On fait bouger le mulot à la prochaine place car on

// lui changera la spécialité que aprè avoir fait bouger

// le grimpeur vers la droite.

Mulot mulot = new Mulot(this.getName(), this.getPosX() + deplacement, this.getPosY(),

Mulot.MulotDirection.EAST, this.getParent());

// On donne l'apparence.

for (BasicCellule cellule : mulot.getListModelisation()) {

cellule.setApparence(mulot.getName().charAt(0));

}

this.changeSpeciality(mulot);

}

}

}

// Si on cherche endehors de la map

catch (ArrayIndexOutOfBoundsException e) {

}

// On fait bouger dans tous les cas le mulot.

finally {

if (this.getDirection() == Mulot.MulotDirection.NORTH) {

this.moveUp(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.SOUTH) {

this.moveDown(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.EAST) {

this.moveRight(deplacement);

} else if (this.getDirection() == Mulot.MulotDirection.WEST) {

this.moveLeft(deplacement);

}

}

}

}

}

MulotParachutiste

package game\_mulot.perso;

import java.util.ArrayList;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicCellule;

import game\_mulot.niveau.Niveau;

/\*\*

\* <p>

\* Mulot qui ne meurt pas d'une très grosse chute.

\* </p>

\*

\*

\*/

public class MulotParachute extends Mulot {

// Constructors.

public MulotParachute() {

super();

this.name = "Parachute";

this.direction = Mulot.MulotDirection.SOUTH;

this.statue = Mulot.MulotStatue.ALIVE;

this.setCurrentAlert(null);

this.fallCounter = 0;

}

public MulotParachute(String name, int posX, int posY, Basic Parent) {

super(name, posX, posY, Parent);

}

public MulotParachute(String name, int posX, int posY, MulotDirection direction, Basic Parent) {

super(name, posX, posY, direction, Parent);

}

// Override methos.

@Override

protected void initializeListModelisation() {

this.listModelisation = new ArrayList<BasicCellule>();

this.listModelisation.add(new MulotCellule(this.getPosX(), this.getPosY(), null, new Character('P'), this));

}

@Override

public void dealAlert(Enum<?> alert) {

MulotCellule.MulotCelluleAlert alertMulot = ((MulotCellule.MulotCelluleAlert) alert);

if (alertMulot == MulotCellule.MulotCelluleAlert.COLLISION) {

if (this.getDirection() == Mulot.MulotDirection.NORTH) {

this.setDirection(Mulot.MulotDirection.SOUTH);

} else if (this.getDirection() == Mulot.MulotDirection.SOUTH) {

// On ne meurt pas quand on tombe.

this.setDirection(Mulot.MulotDirection.EAST);

Mulot mulot = new Mulot(this.getName(), this.getPosX(), this.getPosY(), this.getDirection(), this.getParent());

// On donne l'apparence.

for (BasicCellule cellule : mulot.getListModelisation()) {

cellule.setApparence(mulot.getName().charAt(0));

}

this.changeSpeciality(mulot);

} else if (this.getDirection() == Mulot.MulotDirection.EAST) {

this.setDirection(Mulot.MulotDirection.NORTH);

} else if (this.getDirection() == Mulot.MulotDirection.WEST) {

this.setDirection(Mulot.MulotDirection.EAST);

}

} else if (alertMulot == MulotCellule.MulotCelluleAlert.ARRIVED) {

((Niveau) this.getParent()).setNumberSaveMulot(((Niveau) this.getParent()).getNumberSaveMulot() + 1);

((Niveau) this.getParent()).setScore(((Niveau) this.getParent()).getScore() + 1);

// On le ren dmort pour qu'il n'avance plus.

this.statue = Mulot.MulotStatue.DEAD;

} else if (alertMulot == MulotCellule.MulotCelluleAlert.OUT\_OF\_MAP) {

this.setStatue(Mulot.MulotStatue.DEAD);

}

// On remet l'alerte courante à null.

this.currentAlert = null;

}

}

Personnage

package game\_mulot.perso;

import game\_mulot.basic.Basic;

import game\_mulot.basic.BasicComponent;

/\*\*

\* <p>

\* Représente un personnage. Il hérite de la class BasicObject et à quatre

\* méthodes abstraites en plus :</br>

\* -<strong>moveUp</strong></br>

\* -<strong>moveDown</strong></br>

\* -<strong>moveRight</strong></br>

\* -<strong>moveLeft</strong>

\* </p>

\*

\*

\*/

public abstract class Personage extends BasicComponent {

// Constructors.

public Personage() {

super();

}

public Personage(String name, int posX, int posY, Basic Parent) {

super(name, posX, posY, Parent);

}

// Abstract methods.

/\*\*

\* Déplace le Personage vers le haut.

\*

\* @param deplacement

\* - le nombre de déplacement vers le haut.

\*/

public abstract void moveUp(int deplacement);

/\*\*

\* Déplace le Personage vers le bas.

\*

\* @param deplacement

\* - le nombre de déplacement vers le bas.

\*/

public abstract void moveDown(int deplacement);

/\*\*

\* Déplace le Personage vers la droite.

\*

\* @param deplacement

\* - le nombre de déplacement vers la droite.

\*/

public abstract void moveRight(int deplacement);

/\*\*

\* Déplace le Personage vers la gauche.

\*

\* @param deplacement

\* - le nombre de déplacement vers la gauche.

\*/

public abstract void moveLeft(int deplacement);

}

# Bilan

Au final, nous pensons que notre projet est énormément ouvert pour être continué et peaufiné davantage. Le jeu reste fonctionnel, avec une difficulté que l’on peut modifier grâce au temps de pause entre chaque tour de boucle, qui équivaut donc à la vitesse de déplacement des mulots.

De manière générale, nous sommes très satisfaits de notre application, nous ne voyons que peu d’améliorations à apporter en terme d’algorithmique, c’est seulement avec plus de temps qu’il aurait été possible de faire un jeu beaucoup plus complet.

A nos débuts, nous nous sommes énormément amusés de nos erreurs, lorsque nos marcheurs se mettaient à voler dans notre niveau, à creuser là où bon leur semblaient et à notre grande surprise, nous avons avancé assez rapidement, en bloquant sur peu de choses. Cependant, lorsqu’il nous arrivait de bloquer, nous étions dans l’obligation de demander un œil externe au code, puisque même si on relisait le code et qu’on recommençait la méthode qui causait un souci, sachant que nous étions tous les deux ceux qui l’avaient conçu, nous avions du mal à voir l’erreur sur le coup.

Ce jeu restait tout de même bien plus amusant à coder que celui du Boogle, nous y avons pris plaisir et avons mis plus de cœur à le coder.