le cnam

PROJET JAVA : GRAVITY LP ACSID (2016 - 2017)

ETCHEVERRY FLORIAN - COSTEDOAT PAUL - DJEBALI TAMEUR

Table des matières

1. Introduction

Pour le sujet de notre projet, après de longues réflexions et beaucoup de temps de travail perdu nous avons finalement convenu de concevoir un jeu.

Pour écrire notre code java et réaliser les fonctions attendues nous avons utilisé l'IDE Eclipse avec lequel nous sommes familiers.

N'ayant jamais codé en Java, nous nous sommes appuyé sur les connaissances acquises en cours, des ressources trouvées sur le net et bien entendu sur la Javadoc.

Aussi, l'outil WindowBuilder simplifiant la création des fenêtres nous a permis de gagner du temps.

Pour l'analyse et la conception, nous avons utilisé le logiciel MagicDraw (https://www.nomagic.com/products/magicdraw). Il permet de manipuler l'ensemble des diagrammes UML.

Dans ce document nous allons détailler les principales fonctions de notre application.

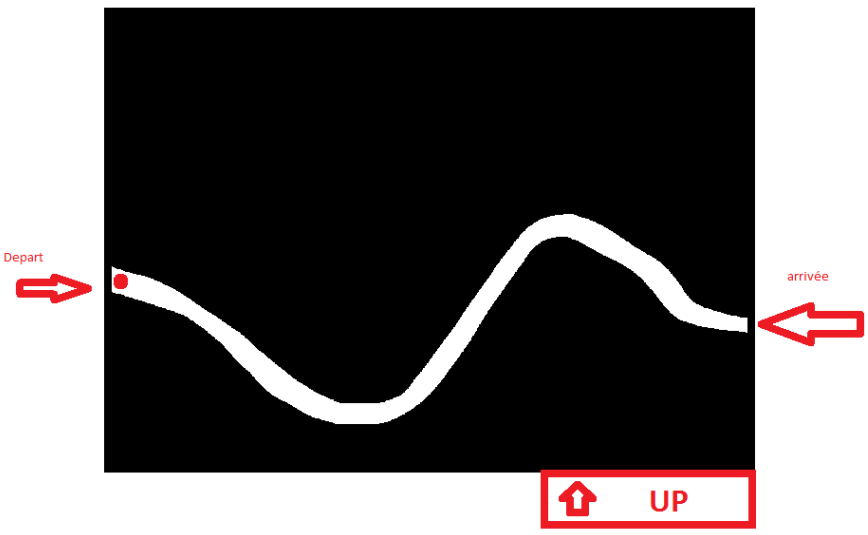
2. Principe du jeu

Le joueur contrôle un objet soumis à la gravité qui doit traverser une fenêtre de gauche à droite en suivant un parcours qui sera de plus en plus difficile selon les niveaux.

L'objet avance vers la droite et tombe vers le bas automatiquement.

Le joueur par l'intermédiaire d'un bouton qui fait monter l'objet pourra ainsi le diriger à travers le parcours sans toucher les bordures qui lui feraient perdre la partie.

2.1. Première idée du visuel



3. Fonctions attendues

3.1. Application

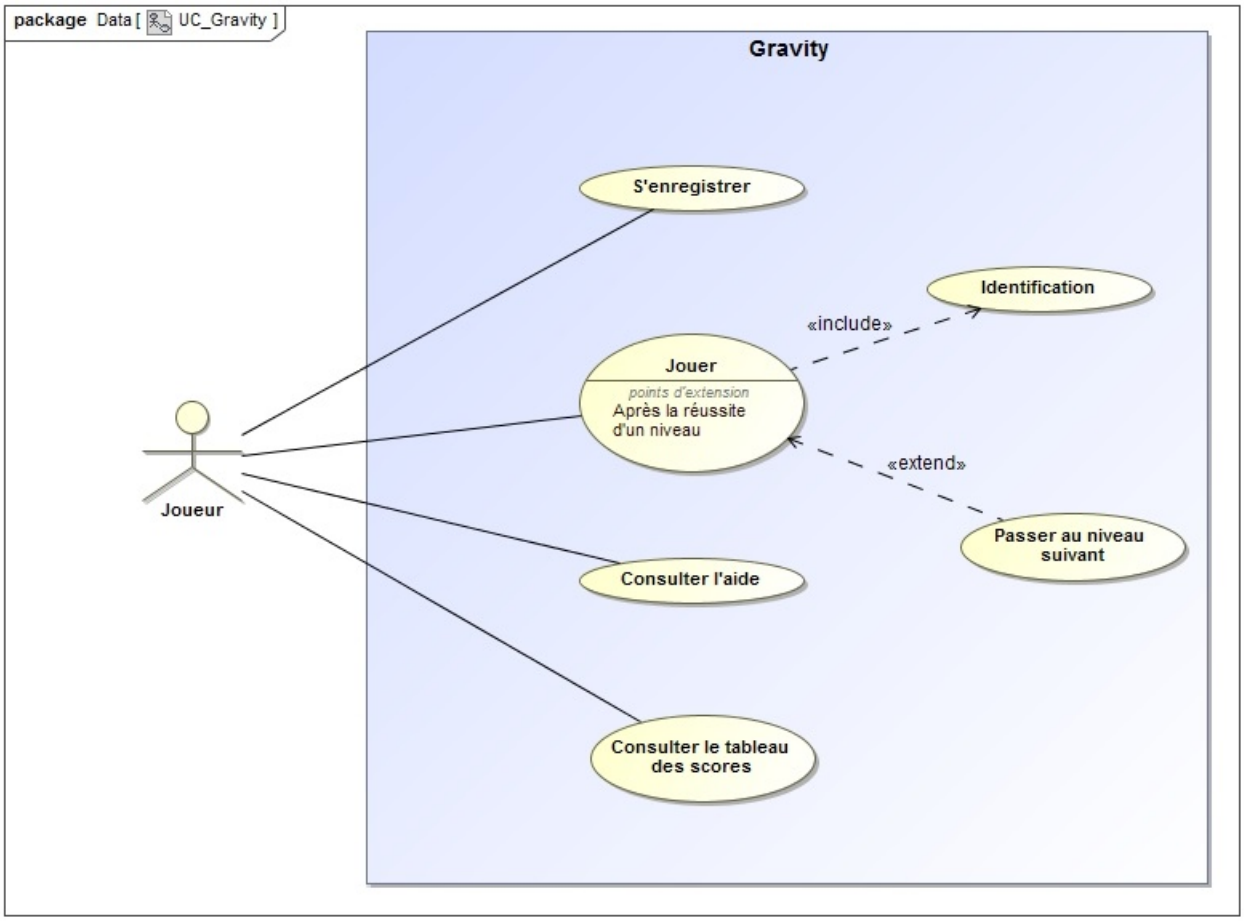
- Plusieurs niveaux de plus en plus difficiles,
- Identification du joueur par un pseudo
- Position de l'objet avance automatiquement sur l'axe X et descend sur l'axe Y
- Bouton UP qui permet de le faire monter sur l'axe Y
- On gagne si on atteint la fin du niveau à droite de la fenêtre et on perd si sort de la zone autorisée
- Reprise du niveau où s'est arrêté le joueur (sauvegarde automatique)
- Tableau des scores
- (Choix des niveaux entre ceux que le joueur a débloqué)

3.2. Interface graphique

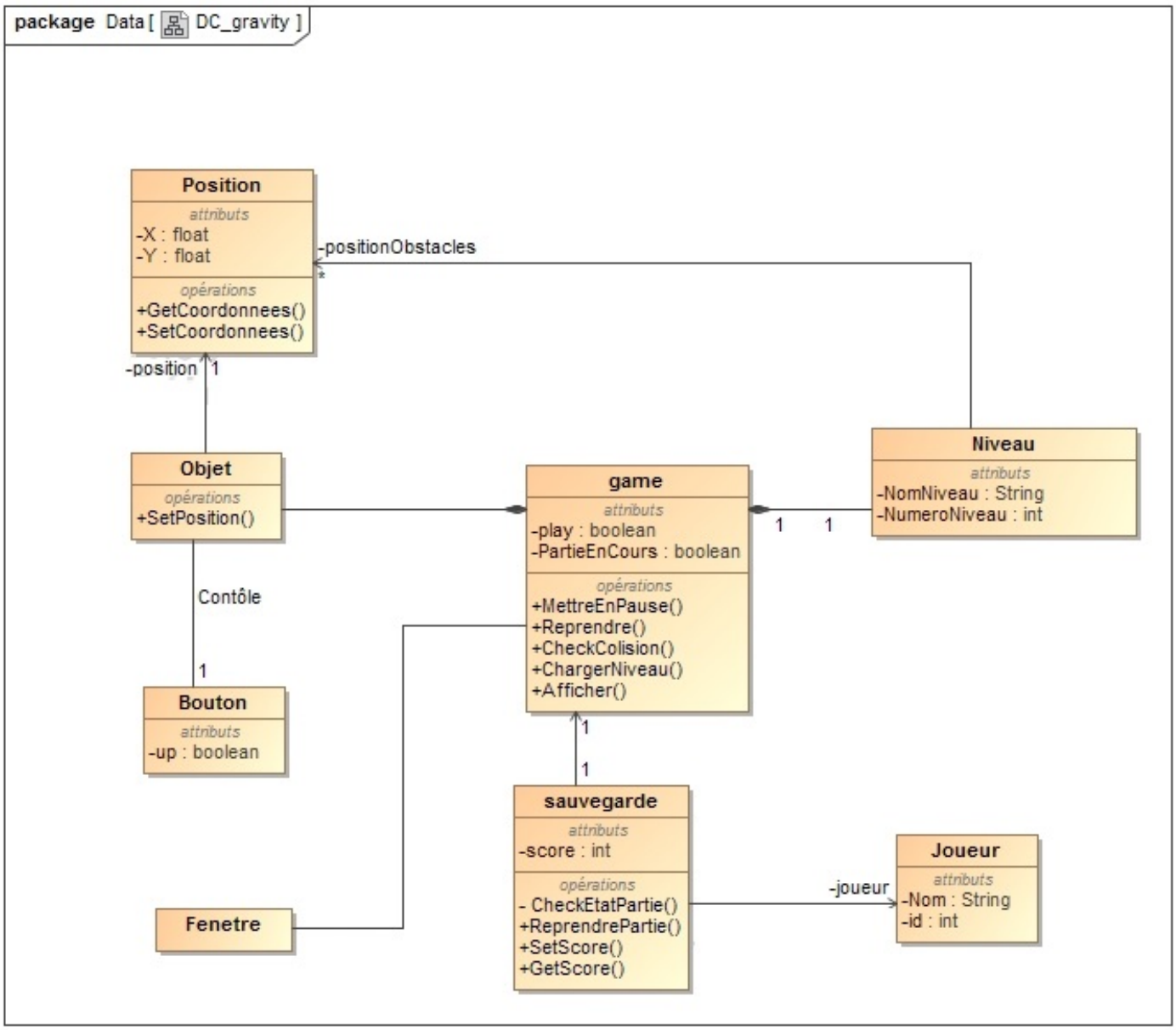
- Informations diverses en haut de la fenêtre (temps, numéro du niveau, nombre d'essais, etc.)
- Zone autorisée où l'objet pourra se déplacer
- Zone interdite où le joueur perd la partie si l'objet la touche
- En bas de la fenêtre le bouton UP

4. Analyse UML

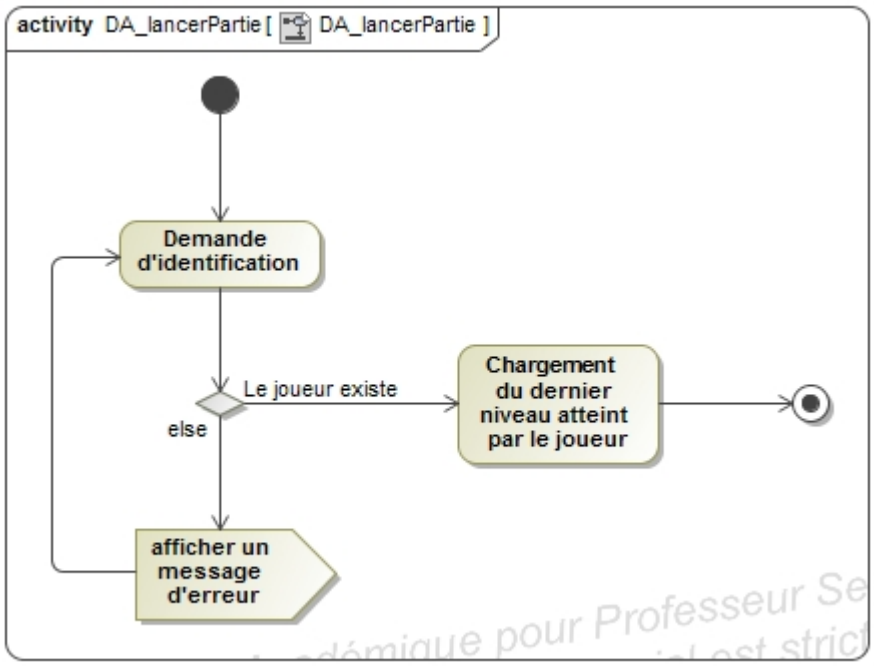
4.1. Diagramme des cas d'utilisation



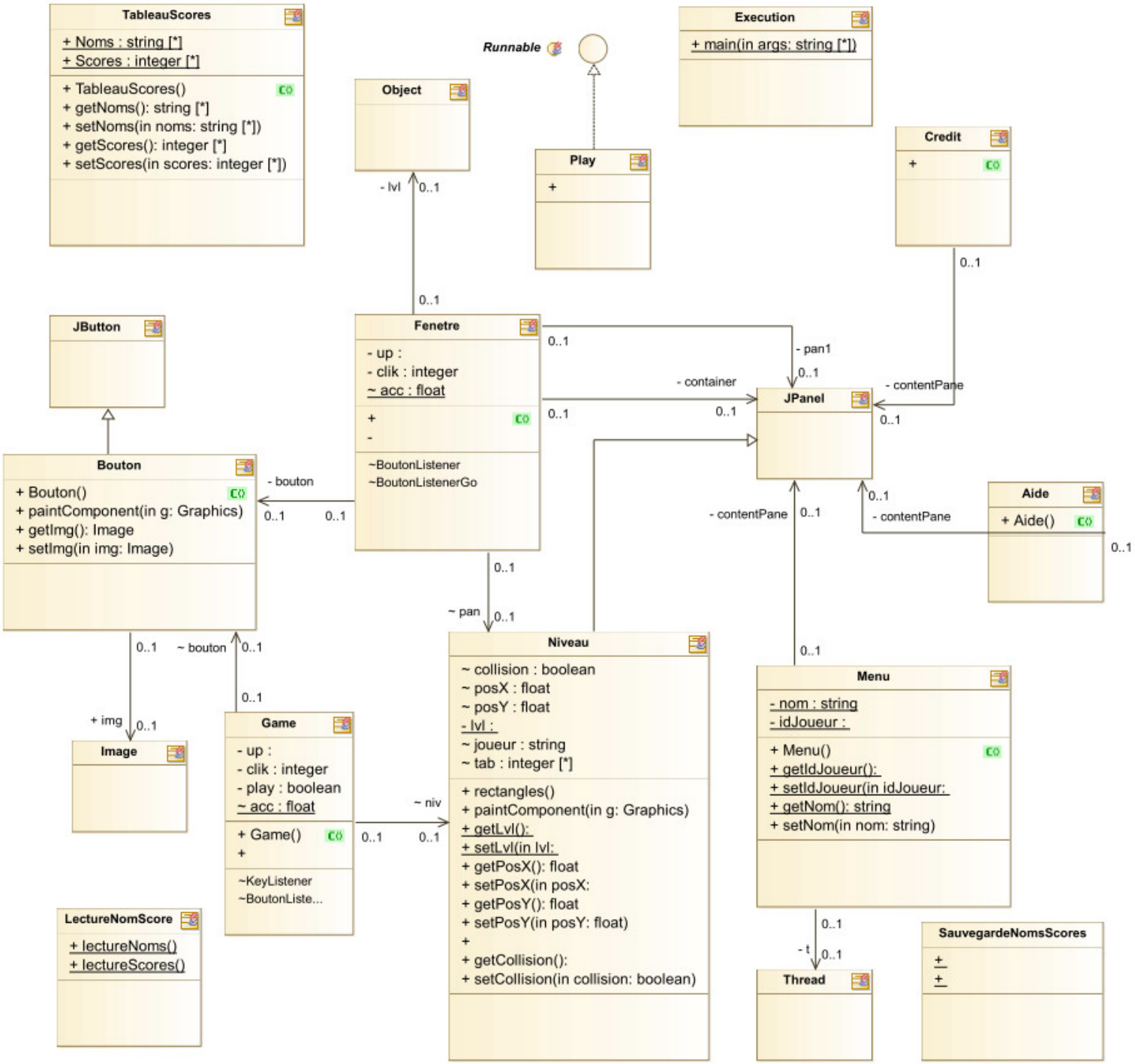
4.2. Diagramme de classes “conception”



4.3. Diagramme d’activité “lancer partie”



4.4. Diagramme de classes “après implémentation”



5. Organisation du travail

Afin de communiquer de manière la plus optimale lors de la phase de développement nous avons utilisé un fichier partagé où étaient reportées les différentes fonctionnalités et les problèmes associés afin de pouvoir les corriger.

Gestion Utilisateur				
	Creation Utilisateur	ok		
Sauvegarde				
	sauvegarde nom	ok		
	sauvegarde score	nok	Retour a la ligne pose pro	
Gestion collision				
	Collision avec bas fenetre	ok		
	Collision avec niveau	ok		
Gestion niveau		ok		
Actualisation Niveau		ok		
Affichage Aide		ok		
Mouvement objet		ok		
Consultation des score		ok		
Lancement Animation		nok	Résolu en utilisant un thr	

6. Détails des fonctions et solutions apportées

6.1. Mouvement de l'objet contrôlé

Rappel : L'objet doit avancer automatiquement sur l'axe X et descend sur l'axe Y. Un bouton doit permettre de le faire monter sur l'axe Y. Il traverse la fenêtre de gauche à droite.

Objet :

Nous avons choisi pour commencer de générer un simple carré de 7 pixels sur 7. Faute de temps pour le travailler c'est ce carré qui est utilisé pour la version rendue.

Nous lui avons attribué deux variables « posX » et « posY » respectivement à 0 et 150 pour définir sa position initiale.

Ci-dessous la partie du code permettant de générer le carré :

```
float posX = 0F;
float posY = 150F;
// Déclaration des deux variables définissant la position initiale du carré

public void paintComponent (Graphics g){

        g.setColor(Color.RED);
        g.fillRect((int)posX, (int)posY, 7, 7);

// Génération d'un carré rouge de 7/7 pixels

}
```

Bouton :

Nous avons ajouté un bouton en bas de la fenêtre. Nous avons aussi créé une classe « Bouton Listener » qui implémente l'interface MouseListener afin de mettre le bouton à l'écoute de la souris et de pouvoir exécuter des actions au clic du bouton.

Nous avons créé une variable « up » initialisée à « false ». Lorsque que l'on clique sur le bouton, « up » passe à « true » et si on le relâche, « up » revient à « false ». L'image du bouton change aussi.

Voici ci-dessous le code du bouton :

```
Bouton bouton = new Bouton();
// Génération d'un bouton
bouton.setBounds(235, 341, 143, 34);
// Positionnement et redimensionnement du bouton
getContentPane().add(bouton);
// Ajout du bouton à la fenêtre
bouton.addMouseListener(new BoutonListener());
// Ajout du Listener au bouton (ici c'est la classe Boutonlistener)
```

Code de la classe BoutonListener :

```
class BoutonListener implements MouseListener {

// Création de la classe qui implémente l'interface MouseListener

        public void mousePressed(MouseEvent event) {
// Les actions suivantes sont exécutées lors du clique sur le bouton

                up = true;
// Passage de « up » à true

                try {
                        bouton.img = ImageIO.read(new File("up1.png"));
// Changement de l'image du bouton

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        public void mouseReleased(MouseEvent event) {
// Les actions suivantes sont exécutées lors du relâchement du bouton

                up = false;
// Passage de « up » à false

                try {
                        bouton.img = ImageIO.read(new File("up.png"));
// Changement de l'image du bouton

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
```

Mouvement :

Nous avons d'abord choisi d'écrire un code simple permettant de contrôler l'objet.

Quand le bouton up n'est pas pressé (up=false) :

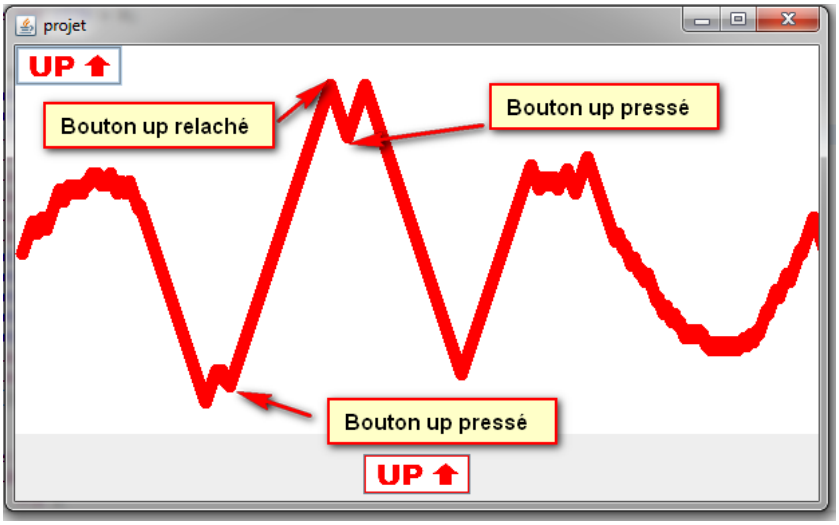
```
Objet.setPosX(Objet.getPosX()+1);
    //L'objet avance d'1 pixel sur l'axe X
Objet.setPosY(Objet.getPosY()+3) ;
    // L'objet avance de 3 pixels sur l'axe Y
```

Quand il est pressé (up=true):

```
Objet.setPosX(Objet.getPosX()+1);
// L'objet avance d'1 pixel sur l'axe X
Objet.setPosY(Objet.getPosY()-3);
// L'objet recule de 3 pixels sur l'axe Y
```

Ceci convenait répondait aux attentes de la fonction.

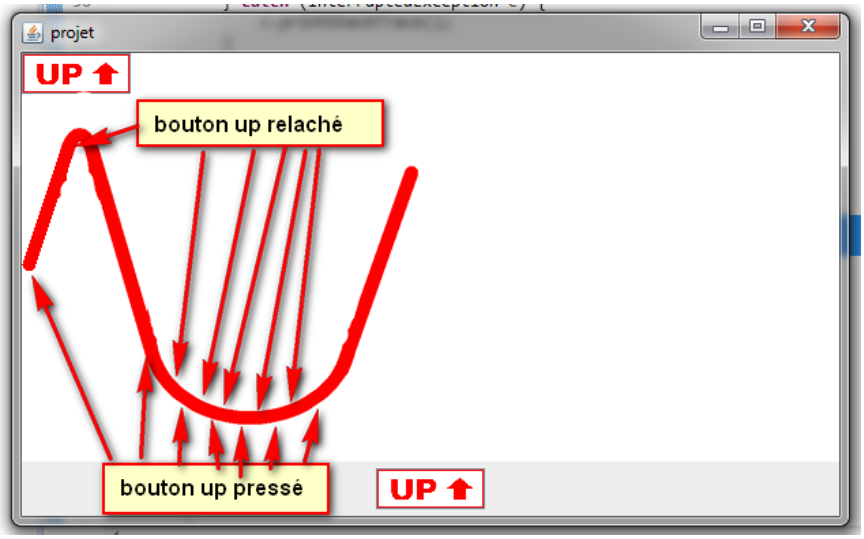
Cependant, comme l'illustre la capture d'écran ci-dessous les déplacements que l'on pouvait réaliser avec le carré n'étaient pas adaptés au types d'obstacles que nous souhaitions lui faire traverser.



En effet seule une trajectoire linéaire pouvait être réalisée par le carré contrôlé.

Pour prendre une courbe il était nécessaire de cliquer à répétition et rapidement sur le bouton.

Il était donc nécessaire de revoir le code du mouvement afin donner un sorte d'inertie à l'objet contrôlé pour qu'il puisse effectuer des trajectoires courbées comme le montre la capture d'écran ci-dessous :



Pour donner cet effet d'inertie sur l'objet on ne devait plus modifier le déplacement du carré lors de l'appui sur le bouton mais plutôt son accélération.

Ce qui nous a permis de travailler sur un code qui convenait :

Quand le bouton up n'est pas pressé (up=false):

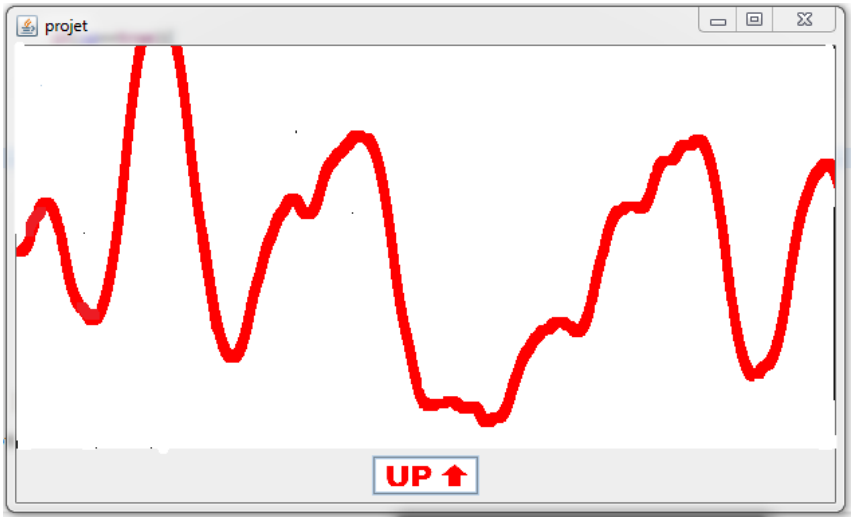
```
acc = acc - 0.15F;
// La variable accélération (acc) est décroissante exponentiellement
niv.setPosX(niv.getPosX() + 0.7) :
// L'objet avance de 0.7 pixel sur l'axe X
niv.setPosY(niv.getPosY() - acc);
// L'objet avance de acc sur l'axe Y
```

Quand il est pressé (up=true):

```
acc = acc + 0.15F;
// La variable accélération (acc) est croissante exponentiellement
niv.setPosX(niv.getPosX() + 0.7);
// L'objet avance d'un pixel sur l'axe X
niv.setPosY(niv.getPosY() - acc);
// L'objet recule de acc sur l'axe Y
```

Ce code fait partie la méthode go() situé dans la classe Game (cf page 23)

Voici le résultat dans la capture d'écran ci-dessous :



Lorsque le bouton est relâché l'objet avance vers la droite et tombe vers le bas à une vitesse exponentielle. Cela donne l'impression qu'il est soumis à une gravité.

Lorsque le bouton est enfoncé l'objet avance vers la droite et tombe vers le haut à une vitesse exponentielle, la gravité est inversée.

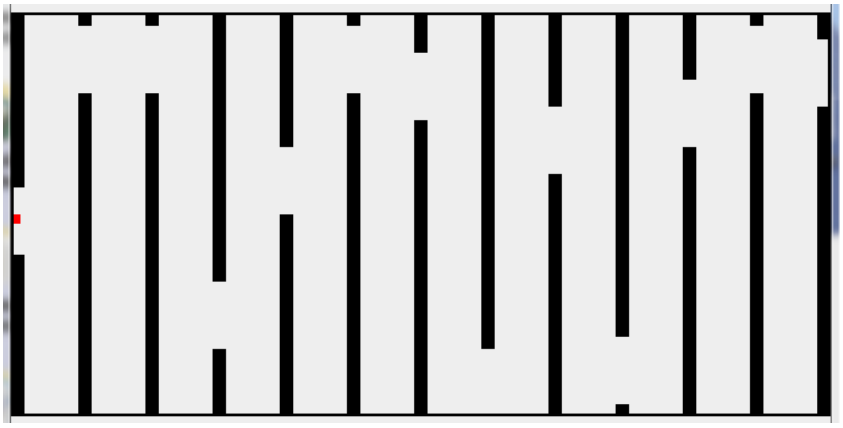
Cette notion de gravité nous a amené à nommer notre jeu « Gravity ».

6.2. Gestion des niveaux

Rappel : L'application doit contenir plusieurs niveaux de plus en plus difficiles.

Stockage :

Afin de simplifier la gestion des niveaux, leur génération et leur stockage. Nous avons choisi de modéliser les obstacles des différents niveaux sous forme de barres comme le montre la capture d'écran ci-dessous :



L'avantage de cette solution est de pouvoir stocker les niveaux dans un tableau et de les générer facilement.

En effet, la hauteur de chaque rectangle supérieur est contenue dans un tableau à deux dimensions. Ainsi, le rang 1 du tableau contient la hauteur des rectangles du premier niveau, le rang 2 du deuxième etc.

Voici, ci-dessous le tableau contenant les 6 premiers niveaux :

```
int tab [][] ={
{130,130,130,130,130,130,130,130,130,130,130,130,130},
//Hauteurs des rectangles supérieurs du niveau 1
{130,150,170,200,150,110,90,80,70,75,90,90,100},//2
{130,100,50,40,100,200,250,250,200,200,100,150,200},//3
{130,200,10,200,100,50,200,250,200,100,50,200,70},//4
{130,10,10,200,100,10,30,250,70,241,50,10,20},//5
{130,250,0,250,0,250,0,250,0,250,0,250,0},//6
};
```

Génération :

Nous avons créé une variable « lvl » qui prend la valeur du niveau du joueur.

Lorsque le niveau est généré, les hauteurs des rectangles supérieurs sont récupérées dans le tableau « tab » en fonction de la variable « lvl ».

Les rectangles inférieurs sont automatiquement espacés de 50 pixels par rapport aux rectangles supérieurs.

Ainsi, par exemple, si le joueur est au niveau 2 la variable « lvl » sera égale à 2 et c'est bien les hauteurs stockées au rang 2 du tableau « tab » qui seront utilisées pour générer le niveau.

Voici ci-dessous le code permettant de générer les 6 premiers rectangles supérieurs et inférieurs :

```
public void paintComponent (Graphics g){

        g.setColor(Color.RED);
        g.fillRect((int) posX, (int) posY, 7, 7);
        //Données du carré rouge

        g.setColor(Color.black);
        //Couleur define en noir
        g.fillRect(0,0,10,tab[lvl][0]);
        g.fillRect(0,tab[lvl][0]+50,10,300);
        //Rectangles de premières positions
        g.fillRect(50,0,10,tab[lvl][1]);
        g.fillRect(50,tab[lvl][1]+50,10,300);
        //Deuxièmes
        g.fillRect(100,0,10,tab[lvl][2]);
        g.fillRect(100,tab[lvl][2]+50,10,300);
        //Troisièmes
        g.fillRect(150,0,10,tab[lvl][3]);
        g.fillRect(150,tab[lvl][3]+50,10,300);
        //Quatrièmes
        g.fillRect(200,0,10,tab[lvl][4]);
        g.fillRect(200,tab[getLvl()][4]+50,10,300);
        //Cinquièmes
        g.fillRect(250,0,10,tab[lvl][5]);
        g.fillRect(250,tab[lvl][5]+50,10,300);
        //Sixièmes
        …etc
}
```

Ainsi, les obstacles en fonction du niveau seront bien générés à chaque appel de la méthode repaint() qui comme son nom l'indique, repeint les composants graphiques de l'objet.

Le carré rouge étant aussi un composant graphique il sera aussi repeint à chaque appel de repaint().

Etant donné que sa position change entre chaque appel de la méthode cela donnera l'illusion du mouvement.

6.3. Gestion des collisions

Rappel : La fenêtre contient une "Zone interdite" où le joueur perd la partie si l'objet la touche. Cette zone est modélisée par les différents obstacles.

Principe :

Le joueur perd la partie dès que le carré touche un des obstacles ou sort de l'espace de jeu. Il était donc nécessaire de vérifier régulièrement, si le carré n'était pas en contact avec un des obstacles ou avec la limite de l'espace jeu.

Pour cela nous avons choisi d'utiliser la classe Rectangle du package java.awt.

Cette classe comporte une méthode « intersect » permettant de savoir si deux objets de la classe rectangle entre en collision.

Il a donc été nécessaire de créer un objet rectangle ayant la même taille et position que le carré rouge et autant d'objet rectangle que de différents obstacles affichés sur niveau en cours.

Génération des rectangles :

Nous avons créé la méthode rectangles() dans la classe niveau qui permet de générer les rectangles à chaque fois qu'elle est exécutée. Ainsi, à chaque fois que le niveau change, la méthode est appelée et les rectangles sont créés en fonction de la position des obstacles.

Ci-dessous la méthode rectangles() :

```
public void rectangles() {
        rect01 = new Rectangle(0, 0, 10, tab[lvl][0]);
        //Génération du rectangle supérieur de première position
        rect02 = new Rectangle(0, tab[lvl][0] + 50, 10, 250);
        //Génération du rectangle inferieur de première position
        rect11 = new Rectangle(50, 0, 10, tab[lvl][1]);
        rect12 = new Rectangle(50, tab[lvl][1] + 50, 10, 250);
        rect21 = new Rectangle(100, 0, 10, tab[lvl][2]);
        rect22 = new Rectangle(100, tab[lvl][2] + 50, 10, 250);
        rect31 = new Rectangle(150, 0, 10, tab[lvl][3]);
        rect32 = new Rectangle(150, tab[lvl][3] + 50, 10, 250);
        rect41 = new Rectangle(200, 0, 10, tab[lvl][4]);
        rect42 = new Rectangle(200, tab[lvl][4] + 50, 10, 250);
        rect51 = new Rectangle(250, 0, 10, tab[lvl][5]);
        rect52 = new Rectangle(250, tab[lvl][5] + 50, 10, 250);
        rect61 = new Rectangle(300, 0, 10, tab[lvl][6]);
        …etc
```

Gestion de la collision :

Nous avons créé la méthode Collision() qui à chaque fois qu'elle est appelée va utiliser la méthode intersects() de la classe rectangle et tester si l'objet rectangle créé qui prend la position du carré rouge est en collision avec les rectangles représentant les obstacles.

Une variable booléenne « collision » a été créée et est passée à « true » s'il y a une collision.

Cette méthode est appelée à chaque changement de position du carré rouge.

Ci-dessous la méthode :

```
public void Collision() {

        Rectangle rect = new Rectangle((int) posX, (int) posY, 7, 7);
                //Génération de l'objet rectangle de même taille et position que le carré
        if ((this.posY > 293 || this.posY < 0 ||
                (rect.intersects(rect01))    ||    (rect.intersects(rect02))    ||    (rect.intersects(rect11))    ||
                (rect.intersects(rect12))    ||    (rect.intersects(rect21))    ||    (rect.intersects(rect22))    ||
                (rect.intersects(rect31))    ||    (rect.intersects(rect32))    ||    (rect.intersects(rect41))    ||
                (rect.intersects(rect42))    ||    (rect.intersects(rect51))    ||    (rect.intersects(rect52))    ||
                (rect.intersects(rect61))    ||    (rect.intersects(rect62))    ||    (rect.intersects(rect71))    ||
                (rect.intersects(rect72))    ||    (rect.intersects(rect81))    ||    (rect.intersects(rect82))    ||
                (rect.intersects(rect91))    ||    (rect.intersects(rect92))    ||    (rect.intersects(rect101))    ||
                (rect.intersects(rect102))    ||    (rect.intersects(rect111))    ||    (rect.intersects(rect112))    ||
                (rect.intersects(rect121))    || (rect.intersects(rect122))){
                //test si le carré rentre en collision avec un obstacle ou si il sort de l'espace de jeu
                        collision = true;
                //si il y a collision la variable "collision" passe à "true"
                }

        }
```

7. Création, identification du joueur et gestion du score

Rappel : On souhaite identifier le joueur par un pseudo, et implicitement lui associer un score.

Principe :

Pour stocker le pseudo des différents joueurs et leurs scores associés nous avons utilisé deux ArrayList, une de chaine de caractère nommée « Noms » et une d'entiers nommée « Scores ».

La position du pseudo dans l'Array sera l'identifiant numérique du joueur. Cet id sera utilisé pour joindre le pseudo du joueur et son score.

Création d'un nouveau joueur et initialisation du score :

Dans la fenêtre du menu principal nous avons ajouté un bouton permettant d'ajouter un nouveau joueur. Une vérification de conformité est effectuée avant l'ajout du nom à l'ArrayList.

Nous avons créé une variable « IdJoueur » permettant de lier le pseudo à son score. Le score d'un nouveau joueur est initialisé à « 1 ».

Par exemple, si le nom d'un joueur est situé en 3ème position de l'ArrayList « Noms » la variable « IdJoueur » va prendre la valeur 3.

Ainsi, si il nécessaire de modifier le score du joueur en cours de partie c'est le score situé en position « 3» de l'ArrayList « Scores » qui sera modifiée.

Ci-dessous le code exécuté lors de l'appui sur le bouton « Nouveau joueur » :

```
JButton btnNewButton_1 = new JButton("Nouveau joueur");
btnNewButton_1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                setNom(JOptionPane.showInputDialog("Entre ton nom"));
                if (TableauScores.Noms.contains(nom) && getNom() != null) {
                        JOptionPane.showMessageDialog(null, "Ce nom existe déja !");
                }
                //Vérification si le nom est déjà utilisé
                if (getNom().isEmpty()) {
                        JOptionPane.showMessageDialog(null, "Ce nom n'est pas correct !");
                }
                //Vérification si le pseudo est correct (n'est pas vide)
                if (!TableauScores.Noms.contains(nom) && !getNom().isEmpty() && getNom() != null) {
                //si les conditions sont respectées
                        TableauScores.Noms.add(nom);
                        //ajout du nom a l'arraylist "noms"
                        setIdJoueur(TableauScores.Noms.indexOf(nom));
                        //initialisation de la variable IdJoueur avec la position du nom dans l'array
                        TableauScores.Scores.add(idJoueur, 1);
```

```
                    //score du joueur mis à 1
                    SauvegardeNomsScores.sauvegardeNoms();
                    //sauvegarde des noms
                    SauvegardeNomsScores.sauvegardeScores();
                    //sauvegarde des scores
                }
        }
});
```

Identification d'un joueur déjà enregistré :

Nous avons ajouté un bouton qui permet au joueur de s'identifier et de reprendre sa partie ou il s'était arrêté et de lancer l'animation.

La variable « nom » s'initialise avec le nom du joueur identifié.

Comme vu précédemment, à l'aide de la variable « IdJoueur », on pourra lui lier son score.

Ci-dessous le code exécuté lors de l'appui sur le bouton « Jouer » :

```
JButton btnNewButton = new JButton("Jouer");
btnNewButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                setNom(JOptionPane.showInputDialog("Entre ton nom"));
                if (!TableauScores.Noms.contains(nom) && getNom() != null) {
                        JOptionPane.showMessageDialog(null, "Ce nom n'existe pas !");
                        // Vérification si le nom existe dans l'ArrayList "Noms"
                if (getNom() != null && TableauScores.Noms.contains(nom)) {
                        setIdJoueur(TableauScores.Noms.indexOf(nom));
                // Initialisation de la variable IdJoueur avec la position du nom dans l'Array
                        Niveau.setLvl(TableauScores.Scores.get(idJoueur));
                        // initialisation de la variable lvl avec le score du joueur
                        t = new Thread(new Play());
                        // Création d'un nouveau thread
                        t.start();
                        // Lancement du thread
                        dispose();
                        // Libère les ressources utilisées par le menu et le ferme
                }
        }
});
```

7.1. Sauvegarde et lecture de la sauvegarde

Rappel : Une sauvegarde automatique doit être effectuée permettant au joueur de retrouver son niveau atteint même après avoir quitté le jeu.

Sauvegarde :

Nous avons créé les fichiers Scores.txt et Noms.txt. Nous avons aussi utilisé la classe FileWirter qui permet d'écrire dans un fichier txt donné en paramètre et la Classe BufferdWriter qui permet de créer une mémoire tampon avant l'écriture sur le fichier.

Nous avons créé une classe SauvegardeNomsScores contenant les méthodes sauvegardeScores() permettant d'inscrire les données de l'ArrayList « Scores » dans le fichier Scores.txt et sauvegardeNoms() pour inscrire les données de l'ArrayList « Noms » dans le fichier Noms.txt

Ces méthodes sont exécutées à chaque modification des ArrayList « Noms » et « Scores ».

Ci-dessous la méthode sauvegardeScores() :

```
public static void sauvegardeScores() {
	FileWriter notreFichier = null;
	BufferedWriter tampon = null;
	// Déclaration des variables utilisées
	try {
		notreFichier = new FileWriter("scores.txt");
		// Déclaration de notre fichier txt ou les données vont être inscrites
		tampon = new BufferedWriter(notreFichier);
		// Création de la mémoire tampon qui prend en paramètre l'objet où les données doivent être écrites
		for (int i = 0; i < TableauScores.Scores.size(); i++) {
			tampon.write(String.valueOf(TableauScores.Scores.get(i)));
			tampon.newLine();
			// Ecrit dans la mémoire tampon les données de l'ArrayListScores en passant à la ligne entre chaque donnée.
			}
	} catch (IOException exception) {
			exception.printStackTrace();
	} finally {
		try {
			tampon.flush();
			// Transfère les données de la mémoire tampon vers le fichier scores.txt
			tampon.close();
			notreFichier.close();
			// fermeture du flux
		} catch (IOException e1) {
			e1.printStackTrace();
		}
	}
}
```

Ci-dessous la méthode sauvegardeNoms() :

```
public static void sauvegardeNoms() {
	FileWriter notreFichier = null;
	BufferedWriter tampon = null;
	// Déclaration des variables utilisées
	try {
		notreFichier = new FileWriter("noms.txt");
		// Déclaration de notre fichier txt ou les données vont être inscrites
		tampon = new BufferedWriter(notreFichier);
		// Création de la mémoire tampon qui prend en parametre l'objet où les données doivent être
		écrites
		for (int i = 0; i < TableauScores.Noms.size(); i++) {
			tampon.write(TableauScores.Noms.get(i));
			tampon.newLine();
			// Ecrit dans la mémoire tampon les données de l'ArrayList Noms en passant à la ligne
			entre chaque donnée.
			}

	} catch (IOException exception) {
			exception.printStackTrace();
	} finally {
		try {
			tampon.flush();
			// Transfère les données de la mémoire tampon vers le fichier noms.txt
			tampon.close();
			notreFichier.close();
			// Fermeture du flux
		} catch (IOException e1) {
			e1.printStackTrace();
		}
	}
}
```

Lecture :

Nous avons créé une classe LectureNomsScores contenant les méthodes lectureScores() permettant de lire les données du fichier Scores.txt et de les écrire dans l'ArrayList « Scores » et lectureNoms() pour lire les données du fichier Noms.txt et de les écrire dans l'ArrayList « Noms »

Ces méthodes sont exécutées au démarrage de l'application. Ainsi les ArrayList « Scores » et « Noms » sont remplies avec la dernière sauvegarde effectuée.

Ci-dessous la méthode lectureNoms() :

```
public static void lectureNoms() {
	FileReader notreFichier = null;
	BufferedReader tampon = null;
	try {
		notreFichier = new FileReader("noms.txt");
```

```
                tampon = new BufferedReader(notreFichier);
                while (true) {
                        String ligne = tampon.readLine();
                        // Lit les lignes de nom.txt et l'inscrit dans la mémoire tampon tant qu'il ya des ligne
                        if (ligne == null)
                                break;
                        TableauScores.Noms.add(ligne);
                        //écrit la ligne dans l'ArrayList "Noms"
                        }
        } catch (IOException exception) {
                        exception.printStackTrace();
        } finally {
                try {
                        tampon.close();
                        notreFichier.close();
                } catch (IOException exception1) {
                                exception1.printStackTrace();
                }
        }
}
```

Ci-dessous la méthode lectureScores():

```
public static void lectureScores() {
        FileReader notreFichier = null;
        BufferedReader tampon = null;
        try {
                notreFichier = new FileReader("scores.txt");
                tampon = new BufferedReader(notreFichier);
                while (true) {
                        String ligne = tampon.readLine();
                        // Lit les lignes de scores.txt et l'inscrit dans la mémoire tampon tant qu'il y a des lignes
                        if (ligne == null)
                                break;
                        TableauScores.Scores.add(ligne);
                        //écrit la ligne dans l'ArrayList "Noms"
                        }
        } catch (IOException exception) {
                        exception.printStackTrace();
        } finally {
                try {
                        tampon.close();
                        notreFichier.close();
                } catch (IOException exception1) {
                                exception1.printStackTrace();
                }
        }
}
```

7.2. Tableau des scores

Rappel : On doit pouvoir afficher un tableau des scores.

Nous avons créé une classe TableauScores qui permet à l'aide des classes JTable et JScrollPane d'afficher dans une fenêtre les données contenues des deux ArrayList « Noms » et « Scores » par l'intermédiaire d'un tableau.

Dans le menu nous avons ajouté un bouton « Tableau des scores » qui permet d'instancier un objet de cette classe.

Ci-dessous le code associé :

```
public TableauScores() {
		super();
		setTitle("Scores");
		setBounds(100, 100, 450, 300);
		setPreferredSize(new Dimension(500, 400));
		setLocationRelativeTo(null);
		setVisible(true);

		Object[][] data = new Object[Noms.size()][2];
		//Création d'un tableau à deux colonnes de taille egale au nombre de joueur enregistré
		String title[] = { "Joueur", "Niveau" };
		//Dénomination des deux colonnes
		for (int i = 0; i < Noms.size(); i++) {
			data[i][0] = Noms.get(i);
			data[i][1] = Scores.get(i);
		//écriture des données des ArrayList "Noms" et "Scores" dans le tableau data
		}
		getContentPane().setLayout(new BorderLayout(0, 0));
		JTable table = new JTable(data, title);
		table.setBounds(206, 346, -173, -341);
		getContentPane().add(new JScrollPane(table));
		//ajout du tableau de scores rempli à la fenêtre

}
```

7.3. Lancement de l'animation

Comme on a pu le voir lorsque l'on clique sur le bouton « Jouer » cela lance l'animation qui est codée dans la méthode go().

Cependant, il a été nécessaire de démarrer cette méthode dans un nouveau thread afin de pouvoir exécuter les tâches simultanément.

Pour cela nous avons créé la classe play ci-dessous :

```
class Play implements Runnable {
	public void run() {
		Game game = new Game();
		//création d'une fenêtre Game qui affiche le niveau
		game.go();
		//appel la méthode qui lance l'animation
	}

}
```

Lors de l'appui sur le bouton « Jouer » on instancie la classe Play dans un nouveau thread :

```
t = new Thread(new Play());
		// Création d'un nouveau thread
		t.start();
		// Lancement du thread
```

L'animation peut fonctionner correctement

Ci-dessous la méthode go() :

```
public void go() {

	JLabel lvlIndic = new JLabel();
	lvlIndic.setText("Niveau: " + Niveau.getLvl());
	//Indication du nom du joueur en haut de la fenêtre
	lvlIndic.setBounds(138, 5, 66, 14);
	getContentPane().add(lvlIndic);
	JLabel joueurIndic = new JLabel("Joueur: " + Menu.getNom());
	//Indication du nom du joueur en haut de la fenêtre
	joueurIndic.setBounds(10, 5, 118, 14);
	getContentPane().add(joueurIndic);

	while (play == true) {
		niv.collision = false;
		//variable collision mise a false
		this.repaint();
		//creation des elements graphiques du niveau
		niv.rectangles();
		//creation des rectangle pour la gestion de la collision
		while (clik == 0 && play == true) {
			try {
```

```
                    bouton.img = ImageIO.read(new File("up.png"));
                } catch (IOException e) {
                    e.printStackTrace();
                }
                while (clik > 0 && niv.collision == false && niv.getPosX() <
niv.getWidth() && niv.getPosY() < 293 && niv.getPosY() > 0) {

                    if (up == true) {
                    //si bouton est enfonçé
                        acc = acc + 0.15F;
                       // La variable accélération (acc) est décroissante exponentiellement
                        niv.setPosX(niv.getPosX() + 0.7F);
                        // L'objet avance de 0.7 pixel sur l'axe X
                        niv.setPosY(niv.getPosY() - acc);
                        // L'objet avance de acc sur l'axe Y
                        niv.Collision();
                        //test si il y a collision
                        } else {
                        //si bouton est relaché
                        acc = acc - 0.15F;
                       // La variable accélération (acc) est croissante exponentiellement
                        niv.setPosX(niv.getPosX() + 0.7F);
                        // L'objet avance de 0.7 pixel sur l'axe X
                        niv.setPosY(niv.getPosY() - acc);
                        // L'objet recule de acc sur l'axe Y
                        niv.Collision();
                        //test si il y a collision
                    }

                    try {
                        Thread.sleep(20);
                        //pause de 20ms
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                    this.repaint();
                }
            }
            if (niv.getPosX() >= niv.getWidth()) {
            //si le carré est sortie de la fenetre du coté droit
                JOptionPane.showMessageDialog(null, "BRAVO!");
                Niveau.setLvl(Niveau.getLvl() + 1);
                //on ajoute 1 au score
                TableauScores.Scores.remove(Menu.getIdJoueur());
                TableauScores.Scores.add(Menu.getIdJoueur(),
                Niveau.getLvl());
                //on l'ecrit dans l'ArrayList "Score"
                SauvegardeNomsScores.sauvegardeNoms();
                SauvegardeNomsScores.sauvegardeScores();
                //on lance la sauvegarde
                lvlIndic.setText("Niveau: " + Niveau.getLvl());
                clik = 0;
                niv.setPosX(0.0F);
                niv.setPosY(150F);
                acc = 0.0F;
                //on reinitialise les variable avec leur valeur de depart
```

```
                    this.repaint();
                    //creation des elements graphiques du niveau
                }

                if (play == false ) {
                    Menu menuprinc = new Menu();
                    //retour au menu principal si la variable play passe à false
                }

                if (niv.getCollision()&& niv.getPosX() < niv.getWidth()){
                    JOptionPane.showMessageDialog(null, "Perdu!");
                    clik = 0;
                    niv.setPosX(0.0F);
                    niv.setPosY(150F);
                    acc = 0.0F;
                    this.repaint();
                }
            }
}
```

8. Conclusion

Ce projet nous a fait de mettre œuvre les connaissances acquises au cours cette année et d'en acquérir de nouvelles. Il nous a permis mieux comprendre certaines subtilités du codage java qui pouvait nous être difficile à concevoir sans pratique.

Nous nous rendons compte que pour de gros projet logiciel l'analyse UML est une étape cruciale pour l'implémentation.

Nous espérons maintenant que notre travail répondra au attente de l'équipe éducative.

Nous n'avons pas ressenti ce projet comme une charge et cela a été un plaisir pour nous de le mettre en œuvre. Nous avions comme idée première de rendre un projet terminé mais pas seulement. Il nous était important que le travail rendu soit à la hauteur de nos attentes au niveau qualitatif.

Nous nous sommes donc beaucoup attardés sur le remaniement du code et son épuration pour parvenir à ce résultat.

Finalement, nous résumons notre projet de la manière suivante : nous avons appris, pratiqué afin de nous perfectionner, le tout de manière organisée !

9. Modifications futures

Optimisation :

Nous avons donc commencé à rechercher ce que nous pouvions améliorer.

Par exemple, à chaque test de collisions, tous les rectangles des obstacles été créé. Nous avons modifié notre code pour créer les rectangles qu'une seule fois au début de chaque nouveau niveau.
Nous avons aussi pu remarquer qu'à chaque mouvement du carré nous rafraichissons toute la fenêtre avec un repaint(). Alors qu'il serait seulement nécessaire de repeindre le fond et le carré mais nous n'avons pas eu le temps de le corriger.

Voici ci-dessous une liste des futures fonctionnalités envisagées :

-Aide plus explicite :
Nous avons remarqué que les personnes à qui nous avons fait tester notre jeu se lançaient sans regarder l'aide et ne comprenais pas le principe.
Nous pensons donc créer un tutoriel pour le joueur qui n'a jamais joué. Un niveau tutoriel permettrait au joueur de comprendre le fonctionnement tout en commençant à jouer.

-Difficulté progressive :
Au-delà du placement des obstacles, l'accélération du carré est aussi un facteur de difficulté.
Après les premiers tests, nous rendant compte que les personnes n'arrivaient pas du tout à jouer tant l'accélération de l'objet était élevée. Nous l'avons diminuée, mais il leur est toujours compliqué de jouer aux premiers essais.
Nous pensons donc augmenter cette accélération en fonction des niveaux.

-Graphisme et son :
Nous aimerions travailler sur les graphismes. Par exemple le carré rouge pourrait devenir un astéroïde. Des sons et des messages aléatoires seraient émis lors des crashs.

-Autres modes de jeu :
Un mode permettrait par exemple de tenter de passer le plus de niveau sans perdre.

-Suppression d'un joueur

-Fichier de sauvegarde crypté ou caché

10. Captures d’écran

Menu principal :

Animation :