Balles en mouvement

Fabien HOARAU, L3 informatique

13 novembre 2018

Résumé

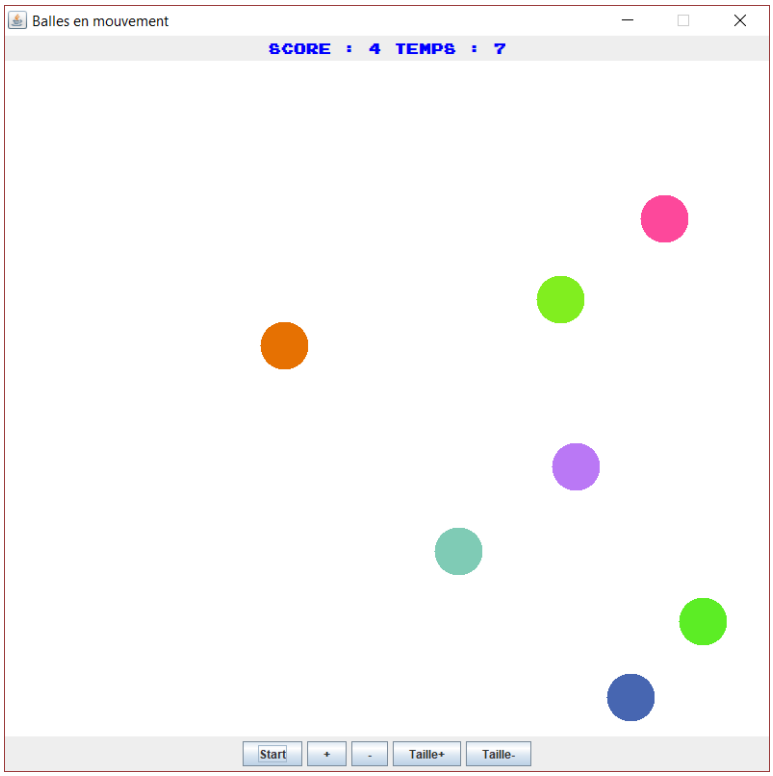
Ce rapport a pour but de présenter l'exercices balles en mouvement le tout réalisé en Java. Je parlerais des différentes méthodes utilisé ainsi que les difficulté rencontrées.

1 Introduction

J'ai choisi de développer le programme sous Java car j'ai plus de familiarité au niveau des librairies graphique par rapport à Python. Le projet se compose de plusieurs classes toutes aussi importantes les unes des autres :

- Balle.java
- Ecran.java
- Fenetre.java
- Main.java
- Panneau.java
- Timer.java
- UI.java

Aperçu final de l'application :



2 Développement

2.1 Présentation : la classe balle

La classe balle représente une grande partie de l'application, sans elle aucune balle ne sera présente. Ici la balle est représentée par des coordonnées *posX* et *posY* et sa taille avec *radius*.

Chaque balles créées sont générées avec des coordonnées aléatoire à l'aide de la fonction *Math.random* Pour les coordonnées aléatoires, la taille de la balle et de la fenêtre sont prises en comptes.

Une balle possède également une couleur elle-même générée aléatoirement. Pour la gestion de la génération de couleur aléatoire, j'ai utilisé la librairie *java.util.Random ;* qui va générer une valeur aléatoire comprise dans un intervalle donnée, ici on fera correspondre ces valeur au code RGB, soit de 0 à 255.
J'ai ensuite initié 3 variables *redValue*,*greenValue* et *blueValue* qui génèrent une valeur entre 0 et 255. Je récupère ensuite ces valeurs pour les appeler lors de l'initiation d'un objet *Color*.

Balle.java

```java
package balles;

import java.awt.Color;

public class Balle {
    int posX, posY, radius, largeur,hauteur;
    boolean backX = false, backY = false;
    Color color;
    Random random = new Random();
    int redValue = random.nextInt(255);
    int greenValue = random.nextInt(255);
    int blueValue = random.nextInt(255);

    public Balle(int fenL, int fenH) {
        this.largeur = fenL + radius;
        this.hauteur = fenH + radius;
        this.radius = 50;
        posX = (int) (radius + (Math.random()) * (fenL - radius + 1));
        posY = (int) (radius + (Math.random()) * (fenH - radius + 1));
        this.color = new Color(redValue, greenValue, blueValue);
        //System.out.println("x :"+ l + "y :"+ h);
    }

    public void paintComponent(Graphics g) {
        Color color = new Color(redValue, greenValue, blueValue);
        g.setColor(Color.white);
        g.fillRect(0, 0, this.largeur, this.hauteur);
        g.setColor(color);
        g.fillOval(this.posX, this.posY,this.radius,this.radius);
    }


    public int getPosX() {return posX;}
    public int getPosY() {return posY;}
    public int getRadius() {return radius;}
    public Color getColor() {return color;}
    public int getTailleX() {return largeur;}
    public int getTailleY() {return hauteur;}
```

Aperçu de la classe Balle, qui gère la structure d'une balle.

2.2 La classe fenêtre

La classe fenêtre s'occupe de placer les différents éléments (boutons, score, couleur de fond,...).

```
public Fenetre() {
this.setTitle("Balles en mouvement");
    this.setSize(800, 800);
    this.setResizable(false);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLocationRelativeTo(null);
    //this.setContentPane(panneau);
    this.setVisible(true);
    ecran.setBackground(Color.white);
    add(ecran);
    add(boutons, BorderLayout.SOUTH);
    add(textes, BorderLayout.NORTH);
    boutons.setLayout(new FlowLayout());
    boutons.add(start);
    boutons.add(plus);
    boutons.add(moins);
    boutons.add(rPlus);
    boutons.add(rMoins);
    //.....
}
```

On peut remarquer que j'ai mis volontairement l'interdiction du redimensionnement de la fenêtre car si on agrandit la fenêtre les balles générées avant agrandissement de la fenêtre resterons dans la zone 800,800 (taille de la fenêtre de départ).
Mais si on agrandit la fenêtre et qu'on ajoute une ou plusieurs balles, ces dernières vont se déplacer dans tout l'espace disponible jusqu'à atteindre un bord ou bien une autre balle.

2.3 Le mouvement et la collision(bordures et balles

```
    public void mouvement() {
        if(posX < 1) {backX = false;}
        if(posX > largeur - radius) {backX = true;}
        if(posY < 1) {backY = false;}
        if(posY > hauteur - radius) {backY = true;}

        if(!backX) {posX++;}
        else { posX--;}
        if(!backY) {posY++;}
        else {posY--;}
    }


    public boolean collision(Balle balle) {
        int d = (this.posX-balle.getPosX())*(this.posX-balle.getPosX()) + (this.posY-balle.getPosY())*(this.posY-balle.getPosY());

        if (d > (radius/25 + balle.getRadius())*(radius/25 + balle.getRadius())) {
            return false;
        } else {
            return true;
        }
    }
}
```

Gestion du mouvement d'une balle et de la collision avec une autre balle.

Mouvement : Le code du mouvement est inspiré de la méthode du site OpenClassroom sur l'exercice du fil rouge dans le cours sur le développement en Java. Le code gère également la collision avec les bordures de la fenêtre. Les booléens *BackX* et *BackY* permettent de tester si l'on recule ou non sur l'axe x et y.

Collision : Pour la collision il s'agit également d'un code provenant d'OpenClassroom depuis le cours sur la Théorie des collisions.
Le principe : un cercle possède un centre x,y et un rayon. Pour savoir si il y a une collision on teste si le centre x,y se trouve dans le cercle. Pour cela, on calcule la distance du point x,y au centre du cercle à l'aide de Pythagore : *d = sqrt((x-C.x)*(x-C.x) + (y-C.y)*(y-C.y))*

2.4 La classe Timer

La classe *Timer* est particulière puisque cette classe gère également le score, lorsqu'on fait appel à *setScore()* on incrémente simplement.(*On fait appelle à setScore() lorsqu'il y a une collision voir le public void run() dans la classe panneau.*

```
public class Timer extends Thread {
    int timer = 0;
    int score = 0;
    boolean state = false;

    public int getTimer() {return timer;}
    public void setTimer(int timer) {this.timer = timer;}

    public int getScore() {return score;}
    public void setScore() {score++;}

    public void setEtat(boolean etat) {state = etat;}
    boolean state = false;
    public void run() {
        while(true) {
        if(state) {
            try {
                timer++;
                sleep(1000);
            } catch(InterruptedException e)
            // TODO Auto-generated catch block
            {e.printStackTrace();}
        } else {
            try {
                sleep(0);
            } catch(InterruptedException e)
            {e.printStackTrace();}
            }
        }
    }
}
```

2.5 La classe UI

La classe UI gère simplement le fait d'afficher le score et le timer dans la fenêtre utilisateur. La classe hérite de JLabel, on peut donc manipuler l'affichage dans la fenêtre grâce aux méthodes :

- **setFont()** : pour définir une police, son caractère et sa taille
- **setText()** : pour pouvoir afficher le score et le temps en récupérant la valeur courante de chacune de ces variables grâce à *timer.getScore()* et *timer.getTimer()*
- **setForeground()** : pour définir une couleur
- **setHorizontalAlignment()** : pour l'alignement horizontal

La déclaration de la variable *timer* du type **Timer** permet de manipuler les variables score et timer de la classe Timer

```
public class UI extends JLabel {
    Timer time;

    public UI(Timer time) {
    this.time = time;
    this.setFont(new Font("Emulogic", Font.BOLD, 12));
    this.setText("Score : " + time.getScore() + "Temps : " + time.getTimer());
    this.setForeground(Color.blue);
    this.setHorizontalAlignment(CENTER);
    }

    public void paintComponent(Graphics g) {
        this.setText("Score : " + time.getScore() + " Temps : " + time.getTimer());
        super.paintComponent(g);
        repaint();
    }
}
```

2.6 Points délicats/intéressants

2.6.1 Points délicats : le mouvement

Le code du mouvement d'OpenClassroom ne fonctionnait que pour une seule balle, il fallait l'adapter pour plusieurs balles, pour cela j'ai utilisé les*ArrayList*.

```
public void run() {
    while(true) {
        if(state) {
            for(Balle balle : balles) {
                balle.mouvement();
                //System.out.println("Largeur : " +balle.getTailleX() + "Hauteur :" + balle.getTailleY());
            }
            for(int i = 0; i < balles.size(); i++) {
                for(int j = i + 1; j < balles.size(); j++) {
                    if(balles.get(i).collision(balles.get(j))) {
                        balles2.add(balles.get(i));
                        balles2.add(balles.get(j));
                        timer.setScore();
                    }
                }
            }
            for(Balle balle: balles2) {
                balles.remove(balle);
            }
            balles2.clear();

            try {
                Thread.sleep(5);
            } catch (InterruptedException e) {e.printStackTrace();}
        }
        System.out.println("panneau state : " +state);
        System.out.println("panneau timer state : " + timer.getTimer());
    }
}
```

L'ensemble des explications données ci-dessous dépendra de la variable booléenne ***state****, si elle est à* ***true*** *on exécute tout.*

Principe : On possède deux listes :

La première liste :

```
ArrayList<Balle> Balles = new ArrayList<Balle>();
```

La deuxième liste :

```
ArrayList<Balle> Balles2 =  new ArrayList<Balle>();
```

La première liste va s'occuper de lister les balles qu'on a ajoutées. En parcourant la liste on vérifie si il y a au moins une balle, si c'est le cas on lance le mouvement grâce à la fonction *mouvement()*.

```
for(Balle balle : balles){
   balle.mouvement();
}
```

La deuxième liste gère les balles qui sont entrées en collisions, si c'est le cas on ajoute ces balles dans cette seconde liste, puis en parcourant la liste on supprime ces balles.

2.6.2 Points délicats : le timer

Pour le timer c'était un problème mineur au niveau d'une variable : Voici le code lorsqu'il y avait un problème :

```
public class Timer extends Thread {
    //...
    boolean state = false;
    public void run() {
        while(true) {
        if(state) {
            try {
                timer++;
                sleep(1000);
            } catch(InterruptedException e)
            // TODO Auto-generated catch block
            {e.printStackTrace();}
            }
        }
    }
}
```

Le problème se trouvait au niveau du *public void run()*, en effet, lorsque je lançais le programme, et que j'appuyais sur le bouton start, le timer se lançait normalement (ainsi que le mouvement des balles) et s'arrêtait lorsque j'appuyais de nouveau. Mais si j'appuyais de nouveau le bouton start, le mouvement se lançait bien mais pas le timer.
Pour résoudre ce problème j'ai simplement rajouté un *else* pour vérifier le cas où la variable *state* est à *false*

```
public class Timer extends Thread {
    //...
    boolean state = false;
    public void run() {
        while(true) {
        if(state) {
            try {
                timer++;
                sleep(1000);
            } catch(InterruptedException e)
            // TODO Auto-generated catch block
            {e.printStackTrace();}
        } else {
            try {
                sleep(0);
            } catch(InterruptedException e)
            {e.printStackTrace();}
            }
        }
    }
}
```

2.6.3 Points délicats : la collision

Un autre point particulier se trouve au niveau de la fonction *collision* de la classe *Balle*.

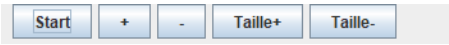
```
public boolean collision(Balle balle) {
    int d = (this.posX-balle.getPosX())*(this.posX-balle.getPosX()) + (this.posY-balle.getPosY())*(this.posY-balle.getPosY());

    if (d > (radius/25 + balle.getRadius())*(radius/25 + balle.getRadius())) {
        return false;
    } else {
        return true;
    }
}
```

On peut voir que j'ai divisé le *radius* par 25. Sans cela les collisions allaient s'effectuer de manière "trop proche".
En divisant le *radius* par une certaine valeur on réduit la taille de la collision pour éviter les collisions trop proches.

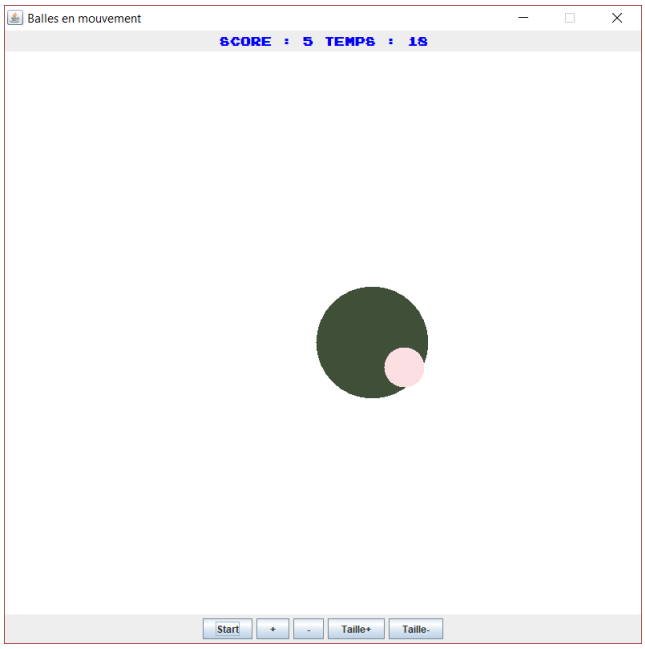
2.6.4 Points intéressants : l'ajout de bouton

Sur le rendu final on peut voir qu'il y a deux boutons qui ont été rajoutés.



Que font-ils ? Ils ont une simple fonctionnalité : il augmente simplement la taille (*radius*) de 10, des balles présentes, lorsqu'on appuie sur le bouton **Taille+** et inversement, on diminue sa taille de 10, si on appuie sur **Taille-**.

Il y a cependant quelque problèmes mineur qui surviennent :
— si la taille d'une balle est supérieure à une autre balle la collision risque d'être faussée.
— si on diminue fortement la taille de la balle, l'utilisateur risque de ne plus voir la balle à l'écran, par contre elle sera toujours en mouvement, suivant l'état du bouton **start**.



Problème de collision si les tailles des balles sont différentes

3 Conclusion

Pour conclure ce rapport, ce projet m'a permis d'étendre plus de connaissances sur le langage java ainsi que les différents procédés et moyens utilisés pour pouvoir réaliser cette application. Il y a encore certaines subtilités, comme par exemple la gestion de la taille. En rajoutant ces fonctions j'ai pu m'apercevoir que la collision ne fonctionne pas complètement lorsqu'on change la taille d'une balle.

Références

[1] Collision. https://openclassrooms.com/fr/courses/1374826-theorie-des-collisions.

[2] Fil rouge. https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java.

[3] Swing. https://docs.oracle.com/javase/tutorial/uiswing/index.html.