TP Programmation Concurrente

Olivier Lebeau, L3 Informatique

12 novembre 2018

Résumé

L'objectif de ce projet consistait à la mise en place d'une interface graphique en Java ou en Python permettant de modéliser un ensemble de balles en mouvement dans une fenêtre.

1 Introduction

Ce projet consistant à la mise en place d'une interface graphique munies de plusieurs caractéristiques distinctes (Animation d'objets, ajout et suppression d'objet, horloge, gestion d'un score) nous avons dans un premier temps, effectué un travail préliminaire afin de définir les différentes classes, threads et objets à mettre en place afin de garantir un fonctionnement optimal de notre programme. Cette décomposition étant l'objet de notre première partie. Nous détaillerons par la suite, l'architecture de notre programme, c'est à dire le contenu et le mode de fonctionnement de ces différentes classes. Enfin, nous terminerons avec les difficultés rencontrées pendant la mise en oeuvre de ce projet et les moyens qu'on pourrait mettre en oeuvre afin d'optimiser et d'améliorer notre programme.

2 Travail préliminaire

Le programme étant composé de différentes fonctionnalités, nous avons dû au préalable décomposés celui-ci en plusieurs classes.

Nous avons choisis pour ce projet le langage Java et avons décomposé le problème sous la forme de plusieurs Threads dont nous détaillerons les spécificités dans une section ultérieure. La décomposition de notre programme s'effectuant de cette manière :

— Une classe pour la fenêtre
— Une classe qui gère l'affichage des balles
— Un objet balle composé des caractéristiques de la balle
— Un Thread qui est chargé de l'horloge
— Un Thread chargé du mouvement et des collisions des balles

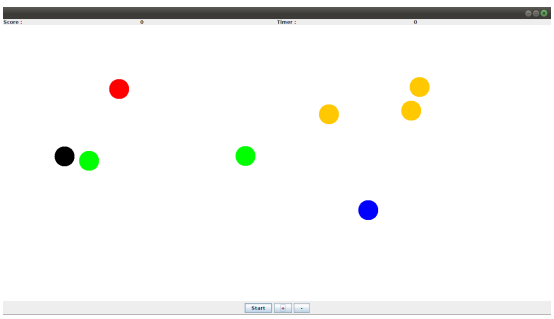

FIGURE 1 – Exemple d'affichage

La figure ci-dessus montre le résultat qu'on obtient une fois toutes nos classes terminés lorsque notre programme est lancé.

3 Architecture du programme

3.1 Les Threads

Notre projet est composé de deux threads (TimerThr et ThrBalles). Le premier est chargé de gérer l'horloge de notre programme tandis que le second gère le mouvement et les collisions éventuelles entre les balles.

3.1.1 L'horloge : TimerThr

La classe TimerThr est un thread implémentant manuellement une horloge. En incrémentant un compteur entre chaque période de sommeil de 1000ms.

```
1     public void run() {
2             try {
3                     while(!isInterrupted()) {
4                             s++;
5                             sleep(1000);
6                             if(pause) {
7                                     synchronized(this) {
8                                             wait();
9                                     }
10                            }
11                    }
12            } catch (InterruptedException e) {
13                    this.interrupt();
14            }
15    }
```

Si l'utilisateur décide de mettre le programme en pause, l'horloge dispose d'une fonction setPause() qui passe la variable pause à true. Le thread se met alors en attente d'une nouvelle action de l'utilisateur pour continuer son exécution.

3.1.2 Le mouvement des balles : ThrBalles

Le thread de gestion du mouvement des balles se base sur le panneau dans lequel les balles évoluent. Le thread se charge d'appeler la fonction nécessaire au déplacement des balles dans un premier temps, puis la fonction permettant la détection des collisions entre les balles.

```
1     public void run() {
2             while(!isInterrupted()) {
3                     balle.moove();
4                     balle.collision();
5
6                     try {
7                             sleep(5);
8                     }catch(InterruptedException e) {}
9                     if(pause) {
10                            try {
11                                    synchronized(this) {
12                                            wait();
13                                    }
14                            } catch (InterruptedException e) {
15                                    e.printStackTrace();
16                            }
17                    }
18            }
19    }
```

De manière analogue à l'horloge, lorsque l'utilisateur décide de mettre en pause le programme, une fonction setPause() est appelée passant la variable pause à true mettant ainsi notre thread en pause jusqu'à ce qu'il y ait une nouvelle intervention de l'utilisateur.

3.2 Objet : Balles

Notre projet est composé d'un seul objet qui constitue la classe Balles. Cet objet ayant pour but de modéliser une balle en la définissant par rapport à sa position x,y mais aussi par sa vitesse vx,vy. Lors de la création d'une balle, la couleur et la vitesse de de celle-ci sont générées aléatoirement.

```
1     public Balles(int x,int y, int size) {
2                 this.x=x;
3                 this.y=y;
4                 this.collision=false;
5                 int vx= (int)((Math.random()*5)+1);
6                 int vy= (int)((Math.random()*5)+1);
7                 this.vx=vx;
8                 this.vy=vy;
9
10                this.size=size;
11
12                Color[] couleur = {Color.orange,Color.red,Color.blue,Color.green,Color.black,Color.PINK
13                int randColor= (int)(Math.random()*(couleur.length-0));
14                this.couleur=couleur[randColor];
15        }
```

3.3 L'interface

3.3.1 La fenêtre : UI

La classe UI est chargée de la mise en page et de la gestion de la fenêtre de notre programme. La mise en page se fait par un BorderLayout avec :

- Au nord : Une barre contenant le décompte du score et de l'horloge
- Au centre : Un objet de notre classe PanBall qui gère l'affichage, le mouvement et les collisions des balles.
- Au sud : Une barre composée d'un bouton "Start" pour lancer les balles lorsque le programme est en pause/ "Stop" pour mettre le programme en pause lorsqu'il est lancé, un bouton "+" pour ajouter des balles avec l'appel à la méthode addBalle() de la classe PanBall, un bouton "-" pour enlever des balles en appelant la méthode removeBalle(boolean var) de la classe PanBalle

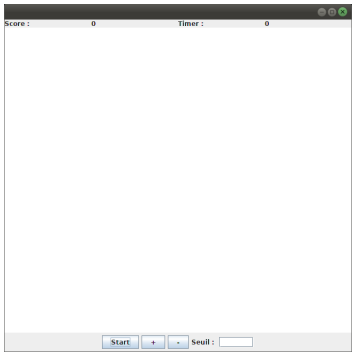

FIGURE 2 – L'interface Graphique

Le bouton "Start"/"Stop" permet à l'utilisateur de mettre le programme en pause ou de relancer celui-ci. Si le programme est en cours, lors d'un clic sur "Stop", les threads en cours se mettent en pause et attendent une nouvelle action de l'utilisateur pour se remettre à fonctionner.

Lorsque le programme est en pause, lors d'un clic sur "Start", la fenêtre avertit les threads leur signalant qu'ils peuvent continuer leur exécution en ayant conserver les informations de l'horloge, du score et la position à laquelle était situés les balles avant la mise en pause. Nous avons aussi ajouté un champ seuil, qui permet à l'utilisateur de spécifier le nombre de balles qui pourront être simultanément dans le panneau.

3.3.2 Le conteneur de balles : PanBall

La classe PanBall est un JPanel au sein duquel les balles vont apparaître et bouger. Elle contient les méthodes permettant le mouvement, la détection de collisions, d'ajout et de retrait de balle. On utilise une liste afin de stocker les balles activent.

La méthode moove() permet la mise en mouvement des balles en plus de détecter lorsqu'une balle arrive sur un bord. Lors du contact avec un bord du panneau, on inverse la vitesse de la balle, ce qui a pour effet de la faire partir dans la direction inverse (rebond).

```
1     public synchronized void moove() {
2               ListIterator<Balles> li= list_Balles.listIterator();
3
4               while(li.hasNext()) {
5
6                       Balles curr= li.next();
7
8                       int x= curr.getX();
9                       int y= curr.getY();
10                      int vx= curr.getVx();
11                      int vy= curr.getVy();
12
13                      if(x < 0)                          curr.setVx(-vx);
14                      if(x+(SIZE) > this.getWidth())     curr.setVx(-vx);
15
16                      if(y < 0 )                         curr.setVy(-vy);
17                      if(y+(SIZE) > this.getHeight())    curr.setVy(-vy);
18
19                      curr.moove();
20
21              }
22
23              repaint();
24
25      }
```

La méthode collision() permet de détecter si deux balles entrent en collision. On utilise un algorithme de détection circulaire, c'est à dire que si la distance entre les deux cercles est inférieure à la somme des rayons de ces cercles alors il y'a une collision.

```
1   public synchronized boolean collision() {
2
3               for(int i =0;i<list_Balles.size();i++){
4                       Balles curr= list_Balles.get(i);
5
6                       for(int j=0;j<list_Balles.size();j++){
7                               Balles curr2= list_Balles.get(j);
8
9                               if(!curr.equals(curr2)) {
10                                      int dx= curr.getX()-curr2.getX();
11                                      int dy= curr.getY()-curr2.getY();
12
13                                      double distance= Math.sqrt(dx*dx+dy*dy);
14
15                                      if(distance < curr.getSize()/2 + curr2.getSize()/2) {
16                                              collisions++;
17                                              curr.setCollision();
18                                              curr2.setCollision();
19                                              return true;
20                                      }
```

```
21                                      }
22
23                              }
24                      }
25                      return false;
26              }
```

La méthode addBalle() permet l'ajout d'une balle en créant un nouvel objet Balles à des coordonnées aléatoire et en l'ajoutant à la liste de balles actives.

La méthode removeBalle(boolean collisions) retire aléatoirement une balle de l'écran lorsque la valeur du paramètre d'entrée est à faux. Si l'argument collision est à vrai la fonction recherche les balles ayant l'attribut collision vrai et les retirent du panneau.

```
1       public void removeBalle(boolean collision) {
2               if(list_Balles.size()>0){
3                       if(!collision) {
4                               int rm_balle= (int) (Math.random()*(list_Balles.size()));
5                               list_Balles.remove(rm_balle);
6                               repaint();
7                       }
8                       else {
9                               for(int i=0;i<list_Balles.size();i++) {
10                                      Balles current=list_Balles.get(i);
11                                      if(current.getCollision()){
12                                              list_Balles.remove(i);
13                                              i--;
14                                      }
15                              }
16                      }
17              }
18      }
```

4 Difficultés et améliorations possibles

La principale difficulté de ce projet venait de la gestion des collisions. En effet, la mise en place de la gestion des collisions conne nous l'avons fais nécessite un certain temps et une certaine consommation des ressources qui peut sans doute être améliorer par le biais d'un algorithme moins gourmant en temps et en ressource.

En effet on est amené à tester par l'algorithme actuelle pour chaque boule si elle est en contact avec toutes les autres boules. Pour remédier à cela, on peut par exemple plutôt que de tester toutes les boules, tester uniquement les boules qui sont dans l'environnement proche de celle considérée via l'utilisation d'un maillage de l'espace. On insérerait une grille qui simulerait notre panneau et nous permettrait de tester uniquement les boules proches.

On peut aussi imaginer un moyen pour que l'utilisateur gère la vitesse à laquelle les boules vont bouger qui est ici régler de manière aléatoire et individuelle lors de la création des boules.

5 Conclusion

Pour finir, on peut dire que le rendu de notre projet constitue une première base qu'on peut améliorer par l'optimisation du code (Notamment la gestion des collisions) mais aussi par l'ajout de fonctionnalités (réglage de la vitesse des balles) afin de rendre l'interface plus efficace et plus interactive avec l'utilisateur.