Rapport

Kevin Jovien, L3 informatique

13 novembre 2018

Résumé

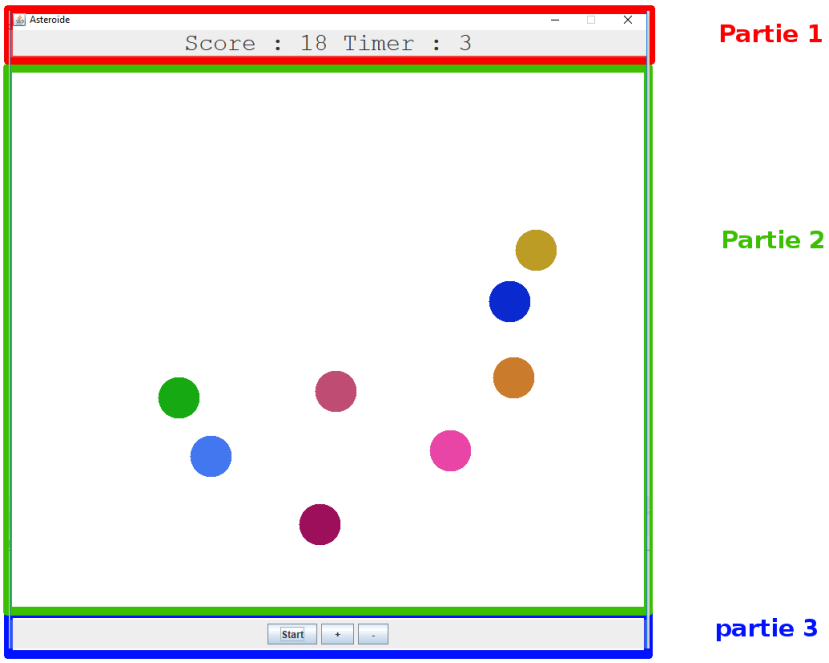
Le présent rapport porte sur la gestion des threads. Pour cela nous allons programmer un mini-jeux de balle. Le jeu possède un tableau des scores qui permet de savoir le temps écoulé et le score, un écran principal qui est tout simplement le jeu et enfin une zone d'interaction avec le jeu . un bouton pause, un bouton qui permet d'ajouter des balles et un qui permet de le retirer. Le tableau des scores et la zone de jeux seront géré par deux threads distincts, pour permettre de gérer aisément la pause simultanée du jeu et de l'horloge.

1 Introduction

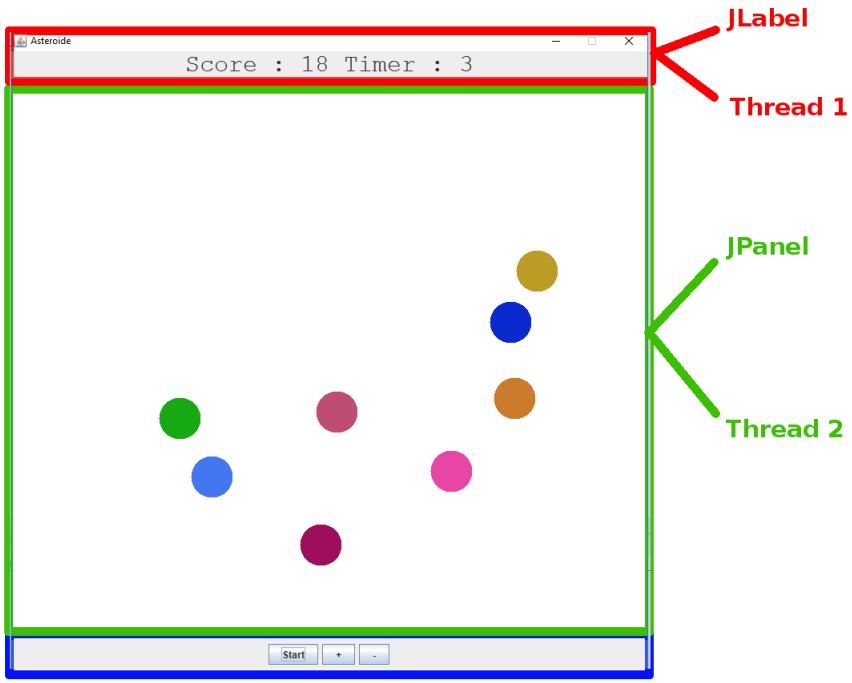
Dans un premier temps on verra le cheminement de la mise en place de l'architecture. Ensuite on verra cette architecture plus en détail et on regardera certains points importants. Puis on soulèvera les points difficiles et les problèmes rencontrent sur le projet. Pour finir, on conclura par une énumération des pistes possibles pour améliorer le programme actuel.

2 Mise en place de l'architecture

Le jeu ci-dessous peut être séparé en 3 parties.



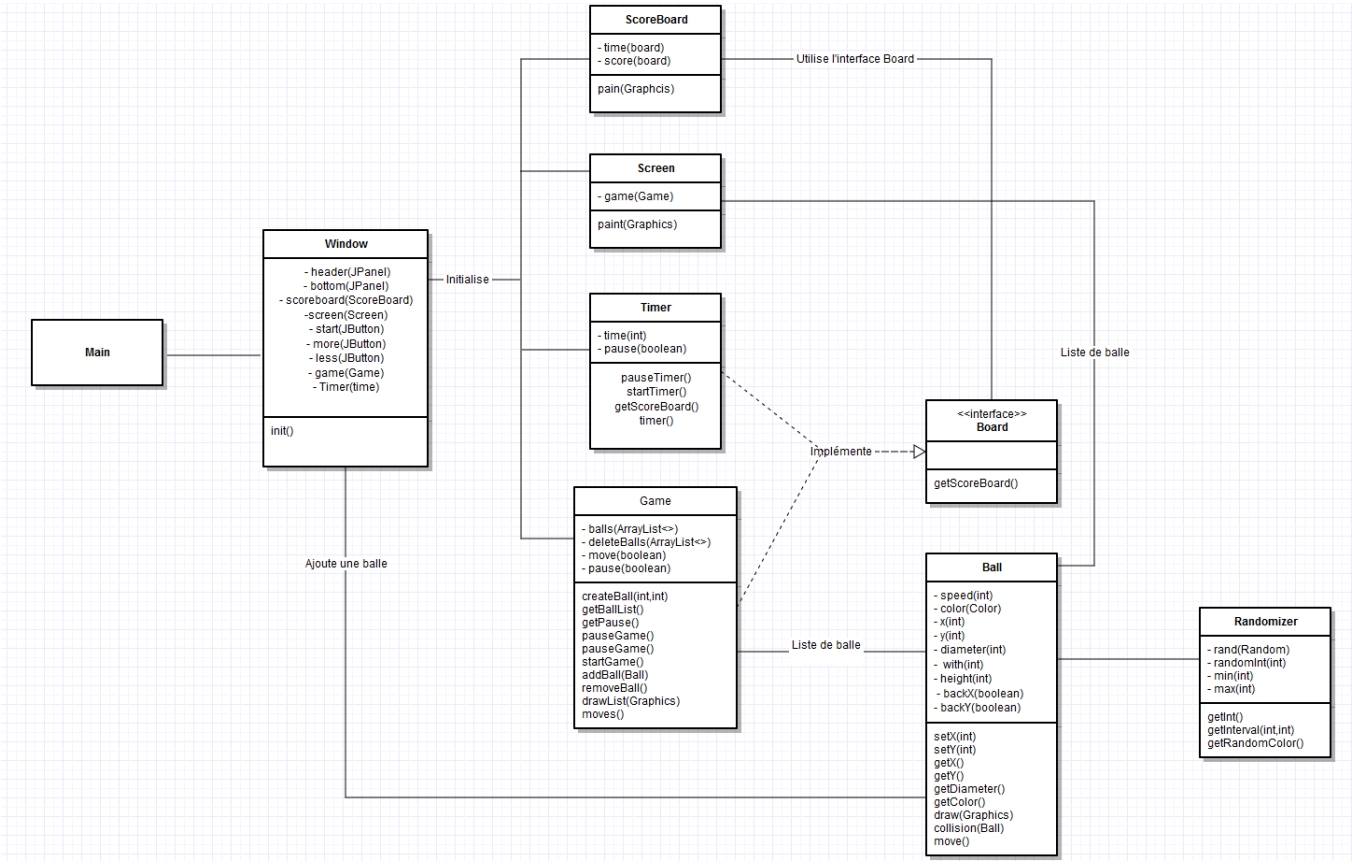
La première et seconde partie du jeu affichent le tableau des scores et le jeu. Ces zones fonctionnent de la même façon qu'un ordinateur. C'est-à-dire un écran qui va n'être que là pour affiche, on va modéliser cela par une classe « JLabel pour le tableau des scores et un « JPanel » pour l'écran de jeu. Et le centre de calcul (processeur, carte-mère, ...) qu'on va ici modéliser par deux threads : une pour le jeu et une pour l'horloge.



L'emploie des deux threads permettra l'implémentation de la fonction pause plus facilement grâce à la méthode *wait()* [3].

3 Achitecture et Code

On va voir maintenant l'architecture du projet et on va expliquer les méthodes utilisées.



3.1 Tableau des scores

Sur le graphe ci-dessus, « l'architecture du projet », on peut voir que les 2 threads Timer et Game implémente l'interface Board. On a créé cette interface pour faciliter l'utilisation du Jlabel Scoreboard. Elle permet de ne pas ce soucier de l'ordre des arguments et facilite l'affichage des résultats. On appelle *repaint* [2] dans la fonction paint pour permettre un rafraîchissement de la Jlabel grâce à la récursivité.

```
public class ScoreBoard extends JLabel{
    Board score,time;
    public ScoreBoard (Board score,Board time) {
        this.score = score;
        this.time = time;
        this.setFont(new Font("Courier New",Font.LAYOUT_NO_LIMIT_CONTEXT,30));
        this.setText(score.getScoreBoard()+" "+time.getScoreBoard());
        this.setHorizontalAlignment(CENTER);
    }

    public void paint(Graphics g) {
        this.setText(score.getScoreBoard()+" "+time.getScoreBoard());
        super.paint(g);
        repaint();
    }
}
```

3.2 Le jeu

3.2.1 Thread Game

C'est dans le thread Game qu'on va gérer la ressource critique, la liste de balles. La liste est manipulée par les méthodes move et drawListe(Graphics). On va donc synchroniser ces méthodes car elle accède à la ressource critique « «liste desballes ». On évite ainsi de parcourir la liste lorsque la fonction move est en train de la modifier (suppression, ajout, etc). Pour rendre l'animation plus naturel, j'ai choisi d'alterner les phases de calcule (ie. les movements de balle) et les phases d'affichages. Pour cela on va utiliser une variable booléen moveball. *verbatim* :

```
public synchronized void drawList(Graphics g) {

    if(!move) {
           for(Ball ball: balls) {
             ball.draw(g);
           }
          move = true;
    }
}

 public synchronized void moves() {

     if(move) {
        try {
          while(pause) {wait();}
        } catch (InterruptedException e1) {
          e1.printStackTrace();
        }
        for(int i = 0 ;i<balls.size();i++) {
            balls.get(i).move();
            for(int j = i+1;j<balls.size();j++ ) {
               if(balls.get(i).collision(balls.get(j))) {
                    deleteBalls.add(balls.get(i));
                    deleteBalls.add(balls.get(j));
                    score = score + 2;
               }
            }
        }
        for(Ball delete:deleteBalls) {
           balls.remove(delete);
        }
```

3.2.2 Le bouton Pause

Pour gérer la pause du jeu et de l'horloge, on va utilise la fonction wait() des threads. Pour cela on utilise une variable booléen pour entrer dans une boucle « tant que » et attendre avec wait. L'utilisation de la boucle est importante car il est possible que le thread mis en attente quitte la file. Pour sortir de la file d'attente on va notifier chaque thread avec notify(). On va les placer dans un bloc synchronisé pour permettre leurs utilisations car la fonction notify ne peut utiliser en dehors d'un bloc synchronisé. Et ainsi permettre de sortir de la pause.

```
start.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
     if(!game.getPause()) {
     game.pauseGame();
```

```
          time.pauseTimer();
          start.setText("Start");
          }else {
          game.startGame();
          time.startTimer();
          synchronized (time) {
          time.notify();
}
          synchronized (game) {
game.notify();
}
          start.setText("Stop");

          }
         }
         });
```

3.3 Problème rencontre et difficultés

L'une des difficultés rencontre a été de savoir quels éléments devrait être gérée par un thread et déterminer la variable critique commune. J'ai donc décidé que l'élément critique serait manipulé que par le thread Game car pour moi la seule variable critique était la liste de balles. Un autre problème rencontré était la gestion de la fonction wait. Car l'utilisation de la fonction notifyall provoque un bug du thread Timer. Pour corriger le bug j'ai donc utilisé un simple *notify* [1] pour cibler le thread important.

3.4 Conclusion

Pour améliorer le jeu, on pourrait ajouter quelques fonctionnalités supplémentaires comme des effets sonores en utilisant un AudioInputStream au moment des collisions entre les balles et les murs. Ou encore permettre aux joueurs de déplacer les balles après une pause aléatoire en utilisant l'event mouse click sur le Jpanel Screen.

Références

[1] Notify. https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#notify().

[2] Repaint. https://docs.oracle.com/javase/7/docs/api/java/awt/Component.html#repaint().

[3] Wait. https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#wait().