# Programmation concurrente - TP

Vincent Fontaine, L3 informatique

9 octobre 2017

**Résumé**

Dans ce rapport, je vous présente les exercices de TP en programmation concurrente dans le cadre de ma formation en L3 informatique à l'Université de la Réunion. Les deux exercices que je présenterai tout au long de ce rapport sont écrits en Python et en Java, en utilisant une interface graphique.

## 1 Première Exercice : File d'entiers

Ici, je reprend l'énoncé de l'exercice :

Écrire un programme qui crée une file d'entiers de capacité maximale 20 ainsi qu'un thread producteur et plusieurs threads consommateurs se partageant cette file et répétant indéfiniment les actions suivantes :

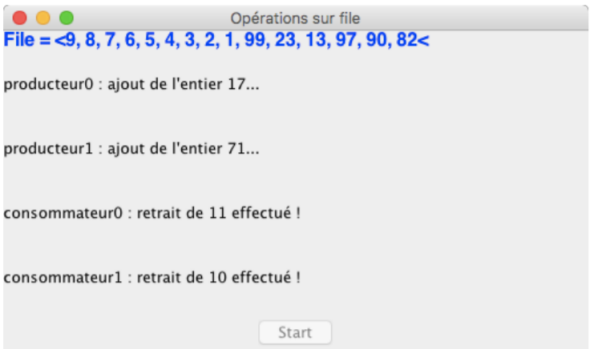
- **Producteur** : Ajout d'un entier choisi aléatoirement entre 1 et 100 puis repos. Si la file est pleine, ce thread est mis en attente jusqu'à ce qu'une place soit disponible.
- **Consommateurs** : Retrait d'un entier, affichage de cet entier puis repos. Si la file est vide, le thread est mis en attente jusqu'à ce qu'un entier soit disponible.

Les temps de repos sont fixés à la création des threads. Les threads sont interrompus lorsque l'utilisateur appuie sur la touche **return**.

Ce programme est écrit en Python.

### 1.1 L'interface graphique

Voici à quoi doit ressembler l'interface graphique pour cette exercice :




Cependant, il n'y a qu'un seul thread relatif au producteur.

### 1.2 Les Threads et les Classes

Les threads utilisés pour la création de cet exercice sont les suivants :

- Producteur
- Consommateur1

— Consommateur2
— Application

Voici donc ce que fait formellement chacun d'eux :

— Producteur : Thread générant un chiffre compris entre 1 et 100 et l'ajoutant à la queue.
— Consommateur1 : Thread retirant un élément dans la queue.
— Consommateur2 : Même que pour Consommateur1
— Application : Classe permettant de créer et gérer la fenêtre TKinter.

### 1.3 Difficultés rencontrées

Les Queues ne permettant pas d'afficher ce qu'elles contiennent, j'ai ainsi créé une liste auquel j'ajoute et retire les même éléments de la queue, ce qui m'as permis d'avoir un retour visuel sur ce que contenait la file.

## 2 Deuxième Exercice : Balles en mouvement

Comme pour la section 1, je commence par rappeler l'énoncé :

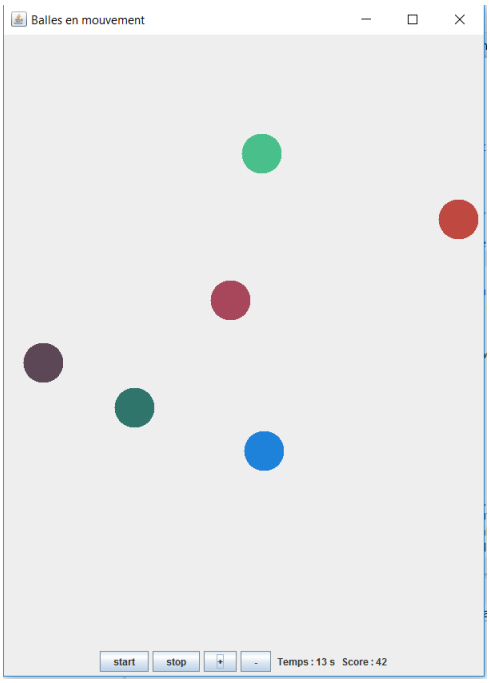
Écrire un programme qui affiche des balles en mouvement dans une fenêtre.

— Les balles ne peuvent sortir du cadre de la fenêtre et ricochent contre les bords.
— Un bouton permet de démarrer/arrêter le mouvement de toutes les balles. Lorsque les balles sont en mouvement, ce bouton a pour étiquette "stop" et lorsque les balles sont à l'arrêt il a pour étiquette "start".
— Un bouton "+" permet d'ajouter une nouvelle balle de couleur aléatoire. Lorsqu'un seuil fixé au préalable est atteint, plus aucune balle ne peut être ajoutée.
— Un bouton "-" permet de supprimer une balle.
— Lorsqu'une collision se produit, les balles impliquées sont supprimées.
— Un score est affiché en permanence et est incrémenté à chaque collision.
— Une horloge affiche le temps écoulé pendant que les balles sont en mouvement ; cette horloge doit stopper son décompte lorque les balles sont arrêtées et reprendre son décompte lorsque les balles sont à nouveau en mouvement.

Cet exercice est écrit en Java.

### 2.1 L'interface graphique

Voici à quoi doit ressembler l'interface graphique de cet exercice :




## 2.2 Les Threads et Classes

Les threads et classes utilisés pour la création de cet exercice sont les suivants :

— Main
— Animation
— Ball
— Horloge
— Fenetre

Voici donc ce que fait formellement chacun d'eux :

— Main : Cette classe sert juste à créer la fenêtre principal en créant un objet Fenetre.
— Animation : Il s'agit du thread gérant les mouvements, les collisions (pour chaque balle) et le score lors des collisions.
— Ball : Cette classe regroupe tous les paramètres de base pour chaque balle.
— Horloge : Un autre thread gérant le temps en seconde.
— Fenetre : C'est le thread principal permettant de créer la fenêtre, possédant les méthodes d'ajout et de suppression des balles.

## 2.3 Les Problèmes rencontrés et parties importantes

— Tout d'abord les problèmes principaux rencontrés se résument à deux méthodes : Le déplacement et les collisions Concernant le déplacement, je me suis limité à un déplacement simple comme suit :

```
public void moveBall()   {
   for(int i=0;i<Bac.length;i++)   {
      if(Bac[i]!=null){
         Bac[i].x = Bac[i].x + Bac[i].speedX;
         Bac[i].y = Bac[i].y + Bac[i].speedY;
      }
   }
}
```

— Puis il y a les collisions, je n'ai finalement pas réussi à faire des collisions balle à balle. Mais les balles ricochent bien sur les bords de la fenêtre.

— Ensuite, il y a le code lancé par le thread de l'animation :

```
public void run(){
   while(!interrupted()){
      this.moveBall();
      this.ballCollision();
      try {
         sleep(10);
      } catch (InterruptedException e) {}
   }
}
```

— Ensuite, il y a le code lancé par le thread de l'animation :

```
public void run(){
while(!interrupted()){
this.moveBall();
this.ballCollision();
try {
sleep(10);
} catch (InterruptedException e) {}
}
}
```

## Conclusion

Ce cours m'a permis de comprendre et d'utiliser les threads en programmation ce qui me permet d'effectuer plusieurs tâches simultanément lorsque je lance un programme. Bien qu'ici l'utilisation des interfaces graphiques et threads dans un même programme m'a fait défaut, j'estime avoir compris comment utiliser un thread.

## Références


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

[2] Tkinter. https://docs.python.org/2/library/tkinter.html.