# Compte rendu du TPL Programmation Orientée Objet :

Ensimag 2A 2019-2020

**Compte Rendu réalisé par :**

Frédéric Alexandre Hayek
Hamza HASSOUNE
Mohamed SABI

# 1 Introduction

Ce sujet traite de plusieurs systèmes multi-agents, différents selon la problématique modélisée, mais présentent tous des similitudes dont nous tirons partie afin d'optimiser notre code.
Dans le rapport qui suit, nous présentons de manière concise notre méthodologie concernant l'implémentation de chaque code visant à résoudre un problème donné, et nous donnons certains détails d'amélioration de celui-ci ainsi que les difficultés rencontrées.

# 2 Un premier simulateur : Jouons à la balle

Q1. Cette première question nous a permis de nous familiariser avec l'environnement du sujet, les outils mis à notre disposition par l'équipe pédagogique afin de réaliser le travail demandé.
Nous avons donc commencer par implémenter cette fameuse classe Balls, et au fur et à mesure des tests effectués, nous avons rajouté des méthodes pertinentes qui nous simplifient les tâches que nous cherchons à effectuer.
Celles-ci sont commentées sur notre code afin de détailler l'utilité et le fonctionnement de chacune d'elle.

Nous avons également pris le soin de généraliser notre code le plus possible, et ce pour pouvoir l'utiliser par la suite et l'adapter afin de résoudre d'autre problématique.
Nous avons pu tester notre code à l'aide d'une classe TestBalls, et le résultats est conforme à ce qui était attendu, la figure ci-dessous l'illuste bien.

```
[java.awt.Point[x=267,y=521], java.awt.Point[x=467,y=253], java.awt.Point[x=573,y=427],
java.awt.Point[x=250,y=547], java.awt.Point[x=8,y=284]]
[java.awt.Point[x=270,y=531], java.awt.Point[x=470,y=263], java.awt.Point[x=576,y=437],
java.awt.Point[x=253,y=557], java.awt.Point[x=11,y=294]]
[java.awt.Point[x=267,y=521], java.awt.Point[x=467,y=253], java.awt.Point[x=573,y=427],
java.awt.Point[x=250,y=547], java.awt.Point[x=8,y=284]]
```

Q2/Q3.Par la suite nous avons eu à représenter graphiquement notre classe Balls, et donc à utiliser le GUISimulator ainsi le GUI.jar afin d'utiliser les classes de figures proposées pour le dessin.

Cf. la figure pour le résultat de la représentation des Balls.

La fonction implémentée dans cette classe et qui sert à initialiser notre fenêtre utilise la fonction random de la bibliothèque Math de Java, afin de générer un nuage de point réparties de façon aléatoire dans le plan.

Q4.Un soin particulier a été apporté à la méthode MettreAJourDirection qui permet d'avoir un rendu plus élégant en simulant des rebondissement sur les parois du cadre (qu'on a représenté graphiquement afin de fixer les parois gauche et inférieure du cadre dans lequel évolue nos balles).

# 3 Des automates cellulaires

On s'attaque dans cette partie à d'autre types de simulation assez classique, consistant à simuler l'évolution dans le temps des états de cellules dans une grille.

## 3.1 Le jeu de la vie de Conway

Q5.Premier essai de simulation sur ce genre de problème : On a pris le soin de nommé les deux états possible d'une cellule pour cette classe.
Un premier essai d'implémentation de la méthode qui permet la mise à jour des états des cellules

de la grille nous a permis de nous familiariser avec ce genre de problème, une fois complétée, on a pu visualiser l'évolution dans le temps d'un ensemble de cellules vivantes.
Notons qu'un déséquilibre a été introduit délibérément dans la méthode permettant d'initialiser les cellules au départ en utilisant la fonction random de la classe Math du langage Java afin de s'arranger pour générer beaucoup plus de cellule morte que de cellules vivantes.

### 3.2 Le jeu de l'immigation

Q6.Au tout début de cette question, on s'est rendu compte que l'on peut généraliser encore plus notre code en créant une classe mère Grille qui tiens compte de tout les paramètres en commun des trois problèmes multi-agents abordés dans cette section, et donc de redéfinir la classe Conway comme sous-classe qui va hériter l'essentiel des méthodes de base. C'est cette même classe mère que nous utiliserons donc afin d'implémenter une nouvelle sous-classe cette fois-ci nommée : Immigration, on tire ainsi partie de tout ce qui a été développé auparavant, et il ne nous reste qu'à redéfinir les méthodes abstraites de la classe mère servant à remettre à jour la grille afin de rendre compte de l'évolution des cellules dans le temps en respectant les algorithme suggérés dans l'énoncé, ainsi que la méthode qui permet d'initialiser l'état de toutes les cellules de la grille à l'instant $t = 0$.

### 3.3 Le modèle de Schelling

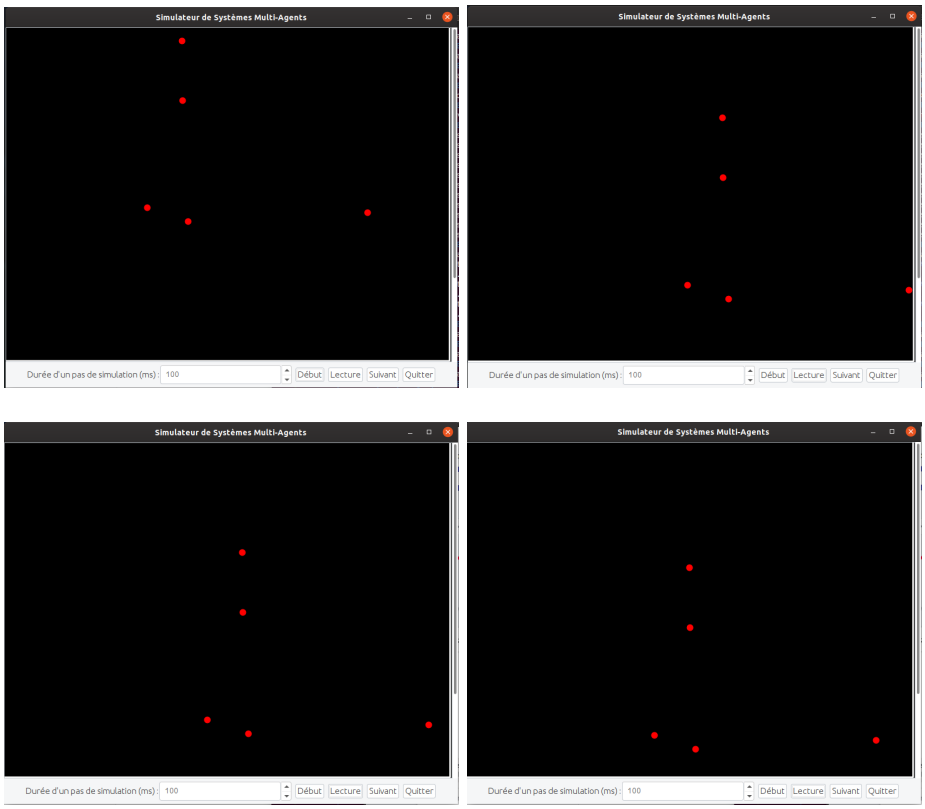
Q7. Le jeu de l'immigration a été simulé en implémentant une sous-classe de notre super classe Grille, la fonction de mise à jour des états des cellules a été spécifiée pour ce test, nous avons pu finalement générer une évolution qui rend compte de l'algorithme proposé par l'énoncé, comme le monte les images ce-dessous.

## 4 Conclusion

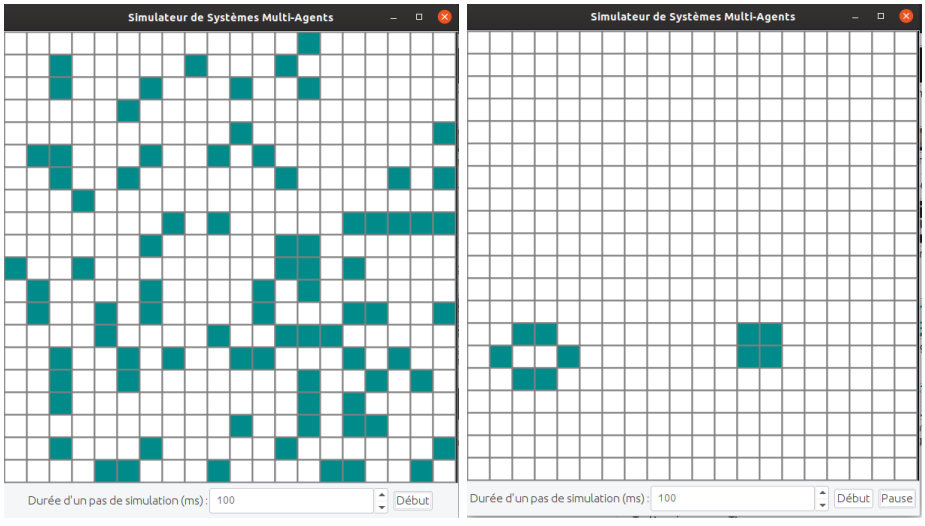
Ce projet a quand même été instructif, malgré les difficultés rencontrées, et même si on a pas eu l'occasion de répondre correctement à toutes les questions posées.
On a appris à gérer plusieurs classes et sous-classes et à tirer profits des similitudes que l'on a remarqué entre les différents systèmes multiagents, ce qui nous a permis de rendre notre code plus lisible, plus organisé vu la factorisation qui s'en est suivie.

# 5 Annexes

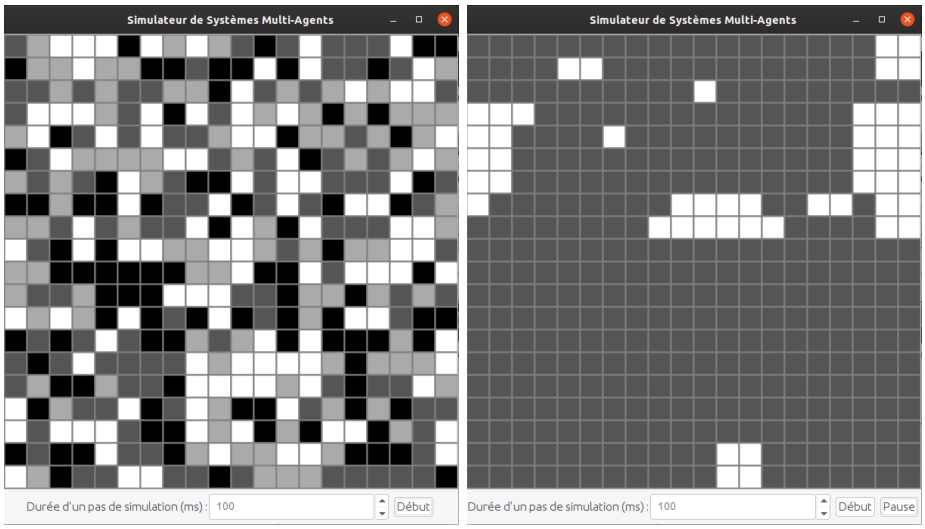
## 5.1 Évolution des balles dans le temps : Illustration du rebondissement




## 5.2 Évolution de l'état des cellules selon le modèle de Conway



### 5.3 Évolution de l'état des cellule selon le modèle de l'immigration




### 5.4 Évolution de la répartition des familles selon le modèle de Schelling