Rapport de TP 4MMPOO : Simulation Orientée-Objet de systèmes multiagents

MAHIEU Lucas (SL_PHELMA)
DEVALON Hugues (SLE_PHELMA)

15 novembre 2015

Préambule L'objectif de ce TP est de développer en Java des applications permettant de simuler de manière graphique des systèmes multiagents. Dans un premier temps, nous nous intéresserons à trois systèmes de type automate cellulaire : le jeu de la vie de Conway, un jeu de l'immigration, et le modèle de ségrégation de Schelling. Dans un second temps, nous nous intéresserons à la simulation d'un système de mouvement d'essaims auto-organisés : le modèle de Boids.
Nous présentons en figure 1 page 1 la représentation UML des classes permettant la Simulation des l'automate cellulaire et des modélisation de *Boids*. Nous avons explicité les attributs privées des classes pour que vous compreniez mieux nos choix de conception et n'avons pas afficher certaine méthode par soucis de clarté.

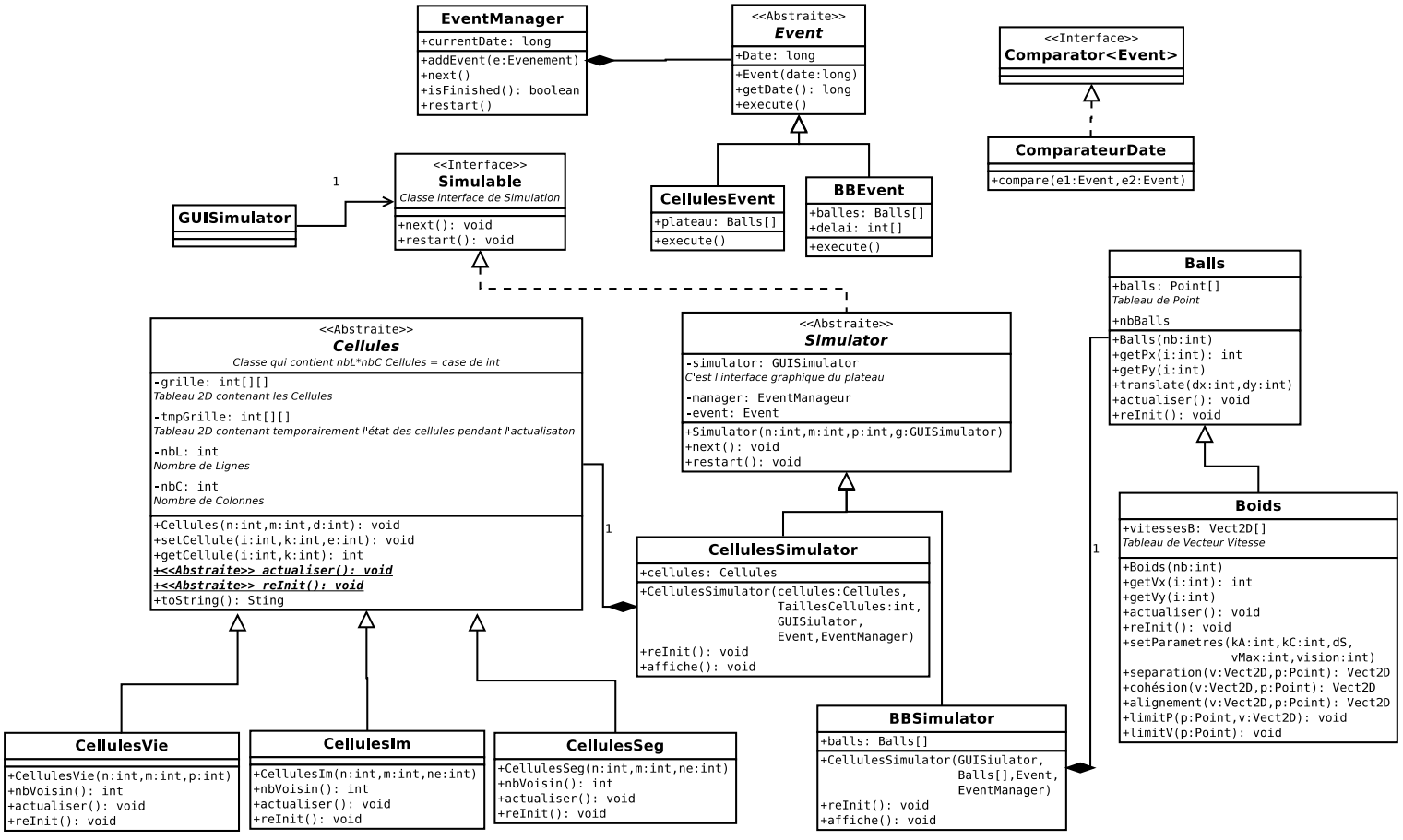

FIGURE 1 – Diagramme des classes du jeu de la vie de Conway

1 Automates Cellulaires

1.1 Jeu de la vie de Conway :

Comme vous pouvez le constater, nous avons utiliser des classes assez 'génériques' pour qu'elles soient facilement réutilisables pour les autres jeux. De plus, on peut remarquer que notre conception est presque identique à celle proposée pour le jeux de balles. En effet, ici nous manipulons un tableau de *int* et non plus un tableau de *Point*. Ainsi pour l'adapter à d'autres jeux, il suffirait de modifier les correspondances entre les états et les couleurs et les règles d'actualisation.

Pour tester ces classes, nous avons utiliser deux classes de tests : l'une pour faire une simulation textuelle dans un premier tant, par la suite, quand la classe de calcule pour tester la simulation graphiquement.

1.2 Jeu de l'immigration :

Les seules différences avec le jeux de la vie sont
— Qu'une cellule peut prendre un nombre n_e d'états et non plus seulement 2.
— Les règles de changement d'états.
Du fait que notre conception est générique, nous avons eu qu'à créer une nouvelle classe fille de *Cellules* qui implement une nouvelles méthode *actualiser()reInit()* propre au jeu de l'immigration.

Nous avons fait le choix de créer une correspondance entre les *Entiers* qui représente l'état d'une cellule et une *Couleur* qui sera affichée dans sa simulation graphique. Pour ce faire nous avons utiliser une table de correspondance *Map<Integer, Color>*.

1.3 Le modèle de Schelling :

Pour cette partie, en plus d'avoir implémenté la nouvelle règle d'actualisation, nous avons utilisé une collection Java pour lister les habitations vacantes : *LinkedList*. De plus, nous avons considéré l'état -1 comme étant l'état d'une habitation vacante pour faciliter les calculs et faisant bien attention que ce nouvel état ne soit pas considéré comme une couleur lors du calcul des voisins. Lorsqu'une famille déménage, elle emménage dans une habitation vacante prise au hasard dans la liste. Lors de l'initialisation, toutes les habitations vacantes prévues sont créées dans l'ordre dans les premières cellules. Ce nombre de cellules vacantes dépend du facteur de ségrégation *k*, du nombre de lignes et du nombre de colonnes, selon la formule : $\frac{lignes*colonnes}{5*(k+1)}$, trouvée empiriquement.

2 Un modèle d'essaims : les *boids*

2.1 Troupeaux de boids

2.1.1 Un gestionnaire à évènements discrets :

Afin de gérer les événements dans *EventManager*, on utilise comme collection Java une *PriorityQueue* afin de classer les événements selon leur date d'execution de façon croissante. La classe *ComparateurDate* implémente cette relation de comparaison. La classe abstraire *Event* et ses sous-classes permettent de définir concrètement la nature des évènements.

2.1.2 Modification de votre simulateur :

Grâce à la hierarchie des classes utilisés, il est facile d'utiliser les événements à la place de l'ancienne gestion. Il suffit alors de créer un sous-événement *BBEvent* pour les balles et les boids et *CellulesEvent* pour tous les types de jeu reposant sur les cellules.

2.1.3 Simulation de plusieurs groupes de boids :

Nous avons généralisé ce problème en proposant aussi de simuler plusieures groupes de balles, les boids étant des balles, cela ne posait pas de problème. Pour cela, il suffit simplement d'utiliser un tableau de balles partout où on les utilise, c'est à dire dans *BBEvent* et dans *BBSimulator* afin d'avoir un simulateur qui contient ces groupes de balles/de boids. Les événements sont alors communs : à la création du groupe on choisit leur délai d'attente entre deux execution, et leur événements s'ajoutent à la queue globale des événements.

3 Bilan