Simulation Orientée-Objet de systèmes multiagents

Année 2021-2022

I. Introduction

Le but de ce TP est d'arriver à simuler des systèmes multi agents basiques : des automates cellulaires et des essaims de "boids". Notre objectif concernant ce TP, en plus de réussir à simuler ces systèmes, était d'obtenir des simulations les plus proches possibles de systèmes réels dans le temps qui nous était imparti.

Tout d'abord, nous avons dû nous organiser pour atteindre cet objectif. Afin de nous organiser le plus facilement possible à distance nous avons décidé de créer un répertoire git, plutôt simplement organisé. A chaque fois qu'une nouvelle partie du projet devait être implémentée, une nouvelle branche était créée en dehors de la branche master. Puis lorsque dans cette nouvelle branche le code avait été testé et validé, il était fusionné à la branche master.

Après s'être rapidement mis d'accord sur l'utilisation de git, la partie la plus importante de notre organisation a donc reposé sur la distribution des tâches. Le sujet se divise en deux parties majeures : la simulation des automates cellulaires et la simulation des "boids," plus complexe. Nous avons donc choisi de suivre le découpage du sujet en affectant une personne sur la première partie et les autres membres du trinômes sur la seconde partie. Cette distribution du travail n'était évidemment pas totalement rigide avec à certains moment des périodes de mise en commun du travail.

II. Conception

Les différentes parties du projet ont été réalisées de manière similaire : on crée une première classe d'**objet simulable** (une balle, une cellule…) qui comporte les données pertinentes pour décrire l'**état courant** et **futur** de l'objet. La classe mère de ces objets est celle que nous avons nommée *SimulableObj*. Ses enfants sont ainsi *Ball* et *Cell* qui représentent une balle et une cellule au sein d'une grille. (voir Fig.1)

Ensuite, nous créons une deuxième classe d'**ensemble d'objets simulables** de même type, fournissant à chaque objet manipulé son état pour la prochaine itération (pour les cellules, c'est une grille qui manipule les objets simulables et choisit dans quel état

seront les cellules en fonction de leurs voisins). Nous avons appelé *SimulableSets* la classe mère de ces objets. Ses enfants sont donc logiquement *Balls*, *Boids* et *Grid* qui représentent respectivement un ensemble de balles, un ensemble de boids et une grille (un ensemble de cellules). (voir Fig.2)

Enfin, on fournit ces ensembles d'objets simulables à un **simulateur** approprié, qui fait le lien entre les **objets manipulés**, l'**event manager** et l'**interface graphique**. C'est lui qui décide des événements à introduire et du nombre d'ensemble d'objets différents. La classe mère de ces objets est la classe Simulator. Ses enfants sont *BallsSimulator*, *GridImmigrationSimulator*, *GridSchellingSimulator* et *BoidsSimulator* qui représentent respectivement un simulateur de balles, un simulateur d'une grille dans le jeu de l'immigration, dans le modèle de Schelling ainsi que le simulateur de boids. (voir Fig.3)

Nous avons alors pu factoriser efficacement notre code avec ces trois familles de classes (*SimulableObj*, *SimulableSets* et *Simulable*).

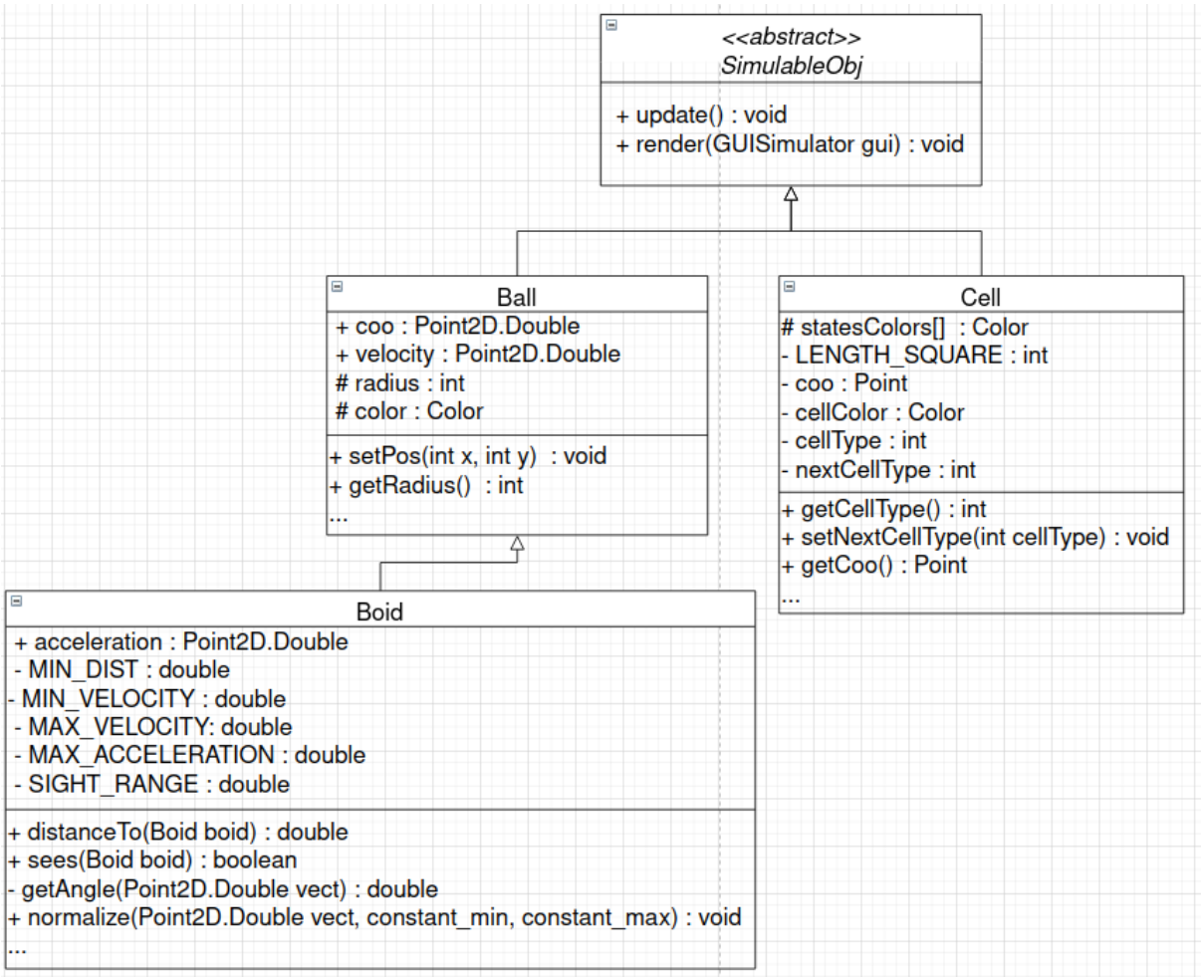

Figure 1 - Diagramme de classes des objets simulables (isolés)

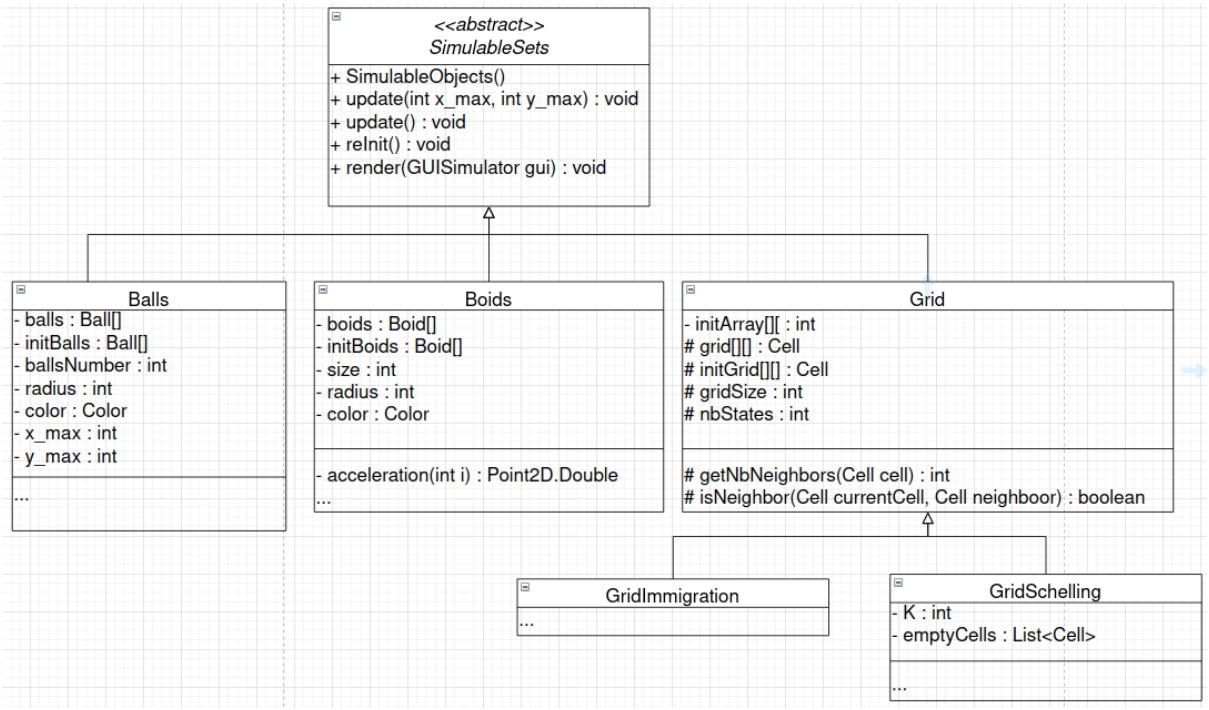

Figure 2 - Diagramme de classes des ensembles d'objets simulables

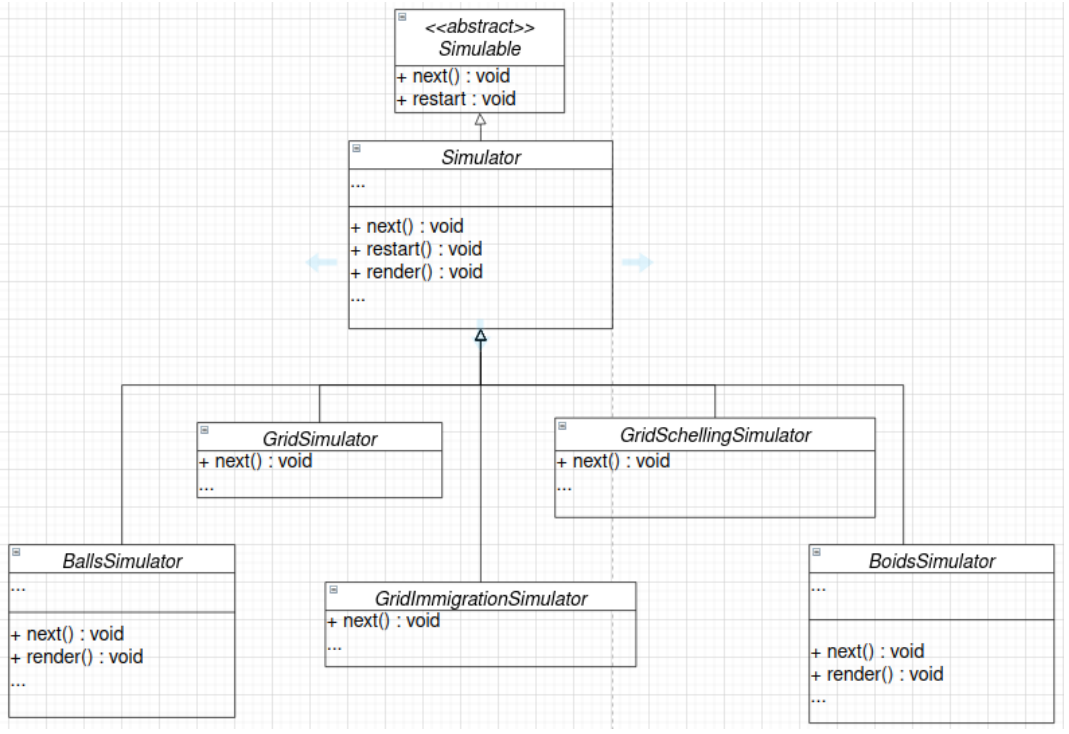

Figure 3 - Diagramme de classes des simulateurs

III. Tests effectués

Balls : la simulation elle-même était le principal test utilisé ; nous avons d'abord fait varier le nombre de balles, puis nous avons ajouté les conditions de bords.

Conway (et autre automates cellulaires) : pour les tests principaux, nous utilisions des grilles 5x5 connues, dont nous connaissions à l'avance l'évolution.

Boids : nous avons effectué un premier test dans lequel nous fixions une cible à atteindre pour un essaim constitué d'un seul boid. Cela nous a permis de comprendre comment utiliser cette cible pour modifier l'état du boid et de déterminer les termes d'attraction et de répulsion, ainsi que d'implémenter le champ de vision d'un boid.

Ensuite, nous avons augmenté le nombre de boids (d'abord deux par essaim) et avons enlevé la cible fixe pour vérifier que le résultat restait cohérent et que la nouvelle cible (le centre de masse des boids dans le champ de vision) était correctement calculée.

Ce dernier test nous a également permis d'observer le comportement des boids lorsqu'on modifiait leur champ de vision, leur vitesse et l'accélération maximale qu'ils sont en mesure de fournir.

Event : nous avons utilisé les tests précédents après avoir implémenté l'*EventManager* et vérifié que le programme fonctionnait toujours de la même manière.

IV. Conclusion

Nous considérons que notre objectif concernant ce projet a été réalisé. Nos simulations des automates cellulaires, que ce soit le jeu de la vie, le jeu de l'immigration ou le modèle de Schelling, ont un comportement cohérent. Nous avons aussi la simulation d'essaims de boids différents gérés par un gestionnaire d'événements, le tout en utilisant un maximum l'héritage afin d'avoir un code plus performant. Cependant avec un peu plus de temps nous aurions souhaité aller plus loin concernant les boids. Nous aurions par exemple aimé modéliser les intéractions qui peuvent exister entre les différents essaims de boids, notamment dans le cas de modèles proie/prédateur ou encore créer des obstacles. Il aurait aussi été possible de faire passer le modèle en 3D en ajoutant une dimension z à nos boids.