Rapport : projet TPL de programmation orientée objet

Rendu le : 19/11/2021

1 Introduction

Le projet consiste à développer une application qui sert à simuler graphiquement des systèmes multi-agents. Ceci se fait par le biais des aspects fondamentaux de la programmation orientée objet, en java. Après avoir réalisé un premier simulateur de balles, les modèles ici développés ne sont autre que Le jeu de la vie Conway, un jeu de migration, le modèle de ségrégation de Schelling et le fameux modèle de Boids.

2 Volet choix de classes et implémentation

Au premier abord, nous avons jugé important de mettre en exergue nos connaissances des principes d'encapsulation et d'abstraction vu la forte utilisation d'une classe dans d'autres fichiers. L'interface graphique (gui.jar) nous a permis de contrôler en soi la simulation des agents et de comprendre leurs mouvements.

2.1 Simulateur de balles

Celui-ci est basé sur l'implémentation de la classe Balls qui est simulée grâce à BallsSimulator. Le Manager des évènements lié à cet exercice est nommé ***nom du fichier ici***, afin de gérer correctement la simulation des divers événements animant le mouvement des balles. Les principaux attributs de la classe Ball désignant le pas et les conditions initiales s'écrivent :

```
private int pas_x;
private int pas_y;
private int xInit;
private int yInit;
```

2.2 Automates cellulaires

Il s'agit de la classe mère, qui contient les différente méthodes commune entre les trois jeux, où chacun de ces jeu hérite de cette classe en question. Cette

classe permet de créer la grille et de déterminer les états à l'aide de la classe Cellule.

```
private int n;
private int m;
private int idAutomate;
protected int reInit;
protected int nbrEtats;
private String[] couleur;
private Cellule[][] grille;
```

Ces attributs sont hérités par les autres classes, qui possèdent des spécificités que l'on gère dépendamment de chaque exercice. Les méthodes exemplaires tels l'initialisation de couleur, la remise en état original et des valeurs avec le processus cyclique sont bien présents dans cette classe mère.

2.3 Modèle des Boids

Ce système permet l'étude de la simulation de déplacement d'agents ici en 2D, les trois règles principales étant respectées (à savoir la cohésion, l'alignement et la séparation). Au point de vue des classes, nous avons implémenté une classe Boid qui décrit le comportement d'un seul boid, quand il interagit avec un second. Ensuite, la classe Boids permet de gérer un ensemble de boids. Celle-ci est fortement utilisée dans Boid.java, puisque le comportement d'un seul boid est indispensablement lié à celui de ses voisins, d'où le choix de raisonner sur la présence ou non des voisins, sur lesquels sont appliquées nos trois règles de simulation.

Les principaux attributs de la classe Boid sont :

```
private float[] positionCourante;
private float[] positionSuivante;
private float[] vitesseCourante;
private float[] vitesseSuivante;
private float[] vitesseOriginal;
private float[] positionOriginal;
private float[] accelerationCourante;
```

Le choix de mettre les attributs en float revient à l'utilisation récurrente des opérations de multiplication et de division par de nombres réels, afin d'éviter les problèmes de l'annulation de certaines valeurs lorsqu'elles sont inférieures à 1 en float. Puisque nous manipulons des forces d'attractions et de répulsions la différence se voit clairement lors de nos simulations.

La gestion des évènements se fait par le biais des classes Event et EventManager. Effectivement, ces deux classes permettent d'affilier les évènements en fonction de l'attribut date qui les ordonnes. Le choix des ListArray comme objet contenant les évènements revient au fait qu'il soit plus fort vis-à-vis à l'implémentation des interfaces.

3 Volet Simulation et algorithmique

3.1 Simulateur de balles

Le simulateur de balles porte sur l'aspect d'un mouvement ordonné de diverses balles, modélisé par un pas en abscisse et en ordonnés. L'arrivée des balles au coin de l'écran nous poussa à implémenter une fonction de rebondissement, qui modélise à cet égard l'équation de mouvement.

Les classes TestBalls.java et dans TestBallsSimulator.java étant des sous classes de TestManager.java, ce dernier simule le tout ! il permet d'avoir une vision claire de ce qui se passe. En effet, nous avons généré divers points que nous translatons à l'aide des méthode de la classe balle, en utilisant l'interface gui.

3.2 Automates cellulaires

Nous avons fait le choix d'implémenter un processus cyclique, traduit par la fonction "int cyclique(,)" dans le fichier AutomatesCellulaires.java. Celui-ci nous a permis de factoriser notre code en quelques lignes de codes, pour ne guère répéter à chaque reprises de nombreuses conditions.

Dans les jeux de la vie Conway, de migration et le modèle de Schelling, nous avons raisonné sur la notion de voisinage, pour qu'on puisse traiter l'état de chaque cellule en fonction de celui des cellules voisines. Les fichiers JeuDeVieSimulator.java, JeuMigrationSimulator.java et ShellingSimu.java sont les aussi des sous classes de TestManager.java qui teste tous les modèles proposées par AutomatesCellulaires. Nous constatons une logique dans nos résultats, basée sur l'implémentation de nos états, avec une vue sur les trois jeux, avec des couleurs de cellules différentes dans chacune des trois interfaces.

3.3 Modèle de Boids

Pour les choix algorithmiques, nous avons choisi d'écrire chaque règle séparément de l'autre, puis de les utiliser lors du parcours de chaque boid, qui est conditionné par l'angle de vision de chaque boid, et de la distance seuil qui le sépare de ses voisins. Pour la règle de séparation, nous avons choisi de diviser celles-ci par le nombre de voisin afin de les équilibrer. Pourtant, nous avons trouvé du mal à déterminer la combinaison parfaite entre ces forces. Pour illustrer ceci, afin de garder la bonne répulsion entre les boids, il sied de diviser par un certain facteur la force de cohésion pour limiter celle-ci lors de nos simulations. Puis, la limitation de vitesse des boids a été nécessaire pour avoir une vision claire de nos objets.

Revenons à l'angle de champs de vision, celui-ci a été implémenté à l'aide

de la formule vectorielle suivante :

$$\theta = arcos(\frac{\vec{U}\vec{V}}{\left\|\vec{U}\right\|\left\|\vec{V}\right\|})$$

Le vecteur $\vec{U}$ étant le vecteur vitesse du boid en question, et $\vec{V}$ le vecteur de liaison entre notre boid et celui que l'on cherche à déterminer son appartenance au champ de vision du premier. Cet angle calculé doit être ainsi inférieur à l'angle de vision commun à tous les boids.

Quant au rebondissement, lorsqu'un boid rencontre un obstacle (ici un mur), il reprend symétriquement son chemin, comme un indiqué dans les simulations ci-dessous. N'omettons pas la cohabitation de plusieurs types de boids, que nous avions réalisée à l'aide de la gestion des évènements.

Le fichier TestManager.java contient tous les tests de simulation chaque jeu du sujet dans son interface, où l'on a inclut dans celui des boids le cas de la présence d'un prédateur et d'une proie. Ceci permet de spécifier le comportement des différents boids selon les différents cas de figure proposés par l'utilisateur. Premièrement, nous avions modélisé le cas où il n'y a que la cohésion, la séparation ou alignement comme règle appliquée. Les boids ordinaires sont modélisés par la couleur noire. Dans le premier cas de figure, modélisé dans une sous-classe de boids nommée BoidsCohesion, si l'on n'applique que la cohésion (couleur rouge), nous observons leur convergence vers le centre de masse. Si nous n'appliquons que la séparation (couleur bleue), il y a divergence mutuelle, à l'aide d'une sous-classe nommée BoidsRepulsifs. Et s'il n'y a que l'alignement (couleur verte), nous apercevons leur respect d'une même direction, et ce, par le biais de la sous-classe nommée BoidsAlignement.

4 Conclusion

En guise de conclusion, les aspects du projets nous ont permis de mieux développer nos compétences en programmation, et de maîtriser la connaissance d'un langage de programmation haut niveau Java, et de la programmation orientée objets. Simuler ces différents modèles met en évidence notre avancement et notre compréhension du sujet.