Rapport Projet

Équipe 58 en Teide

Introduction :

Dans ce document on expliquera nos choix de conception dans chaque partie du TP et on va décrire les différentes méthodes et classes les plus adaptées qu'on a utilisé et on notera à la fin les tests effectués et les résultats obtenus.

I)- Partie Balles :

1) Description des fichiers java :

Point.java : C'est un fichier qui hérite le package existant « java.awt.Point » . Son fonctionnement principal est le suivant :

-Sauvegarde des coordonnées originales de chaque balle sur un ensemble de balles.

-Affichage des coordonnées de la manière suivante : (x,y) avec x et y respectivement l'abscisse et l'ordonnée de la balle.

Balls.java : Contient la déclaration sur la liste des points et contient les méthodes suivantes :

-ajoutePoint() : Permet d'ajouter un point dans la liste avec des coordonnées générés aléatoirement en utilisant le package random().

- bounce() : Prenant en paramètre un point p qui permet de rebondir les balles en faisant des test sur le dépassement des coordonnées du point de la taille de la fenêtre.

-translate(): Permet la translation des différente points présentes dans la liste en faisant des rebondissement.

-reInit(): Réinitialisation de la positions des points présents dans la liste.

-toString(): Permet d'afficher tous les points qui sont dans la liste.

BallSimulator.java : Implémenter le Gui.Simulable en faisant un Override sur les méthodes next() et restart().

BallsEvent.java : Hérite de l'abstract class Event et contient les deux méthodes execute() et reInit() qui ont pour rôle de faire des traitements sur les balles lorsqu'un évènement est arrivé.

Eventmanager.java : Permet le managing des évènements dans une liste et génère un Ballevent à la date choisie._

2) Tests effectués :

On a créé 4 différents fichiers pour faire des tests séparément :

« 1LesBalles » : Translation des balles.

« 2Affiche_Sans_graphique1 » : On génère aléatoirement un grand nombre de points et on effectue des translations constantes.

« 3Affiche_en_graphique » : Chaque appui sur le bouton « Suivant » anime les translations des points.

« 4Insersion_Evenement » : Les translation des balles sont effectués à un certain temps/date.

Il faut utiliser le MAKEFILE pour tous les effectuer tous les tests.

Pour tout test, compile avec 'make' et exécuter avec 'make exeGUI '.

II)- Partie automates cellulaires :

1)- Description des fichiers java :

Cellule.java : une classe qui gère les coordonnées, les dimensions et l'état de la cellule. Pour les trois jeux, le constructeur de la cellule admet comme attributs l'état précédant et courant afin de mémoriser l'état de la cellule à t et t+1 pour éviter des résultats incohérents lors du changement de la grille. Pour **le jeu de l'immigration** on ajoute l'attribut « nbrEtats » qui précise le nombre d'états que peut avoir une cellule. Pour **le modèle de Schelling** on ajoute les attributs « couleurCellule » et « couleurPrecedante » qui représentent l'état de la cellule.

Méthodes du **Jeu de La vie** :

- etat() : retourne l'état de la cellule pour l'examiner

-celluleVivante() : changer l'état de la cellule de morte à vivante

- memoriser() : mémoriser l'état de la cellule à l'instant t

-changer(int nbrVoisins) : changer l'état de la cellule en fonction du nombre de cellules vivantes passé en paramètre

Méthodes du **Jeu de L'immigration** :

- etat() : retourne l'état de la cellule pour l'examiner

- memoriser() : mémoriser l'état de la cellule à l'instant t

-changer(int nbrVoisins) : changer l'état k de la cellule en fonction du nombre de cellules dans l'état k+1

Méthodes du **Modèle de Schelling** :

-setCouleur(int couleur) : changer la couleur de la cellule

-testcelluleVacante() : retourne un booléen qui teste si la cellule est vacante ou non

- memoriser() : mémoriser l'état de la cellule à l'instant t

-testCouleur(int couleur) :teste si la couleur en paramètre est différente de la couleur de la cellule

Grille.java : une classe qui représente la grille de cellule , a comme attributs les dimensions de la fenêtre ,le nombre de lignes et de colonnes et la matrice dont les cases sont de type cellule. Pour le **Modèle de Schelling** on ajoute comme attribut une liste chaînée qui contient les cellules vacantes disponibles.

Méthodes du **Jeu de La vie** et du **Jeu de L'immigration** et du **Modèle de Schelling** :

-ajoutCellules() : initialiser la grille

-mémoriserGrille() : mémoriser tous les cellules de la grille

-changerGrille() : parcourir la grille , pour chaque cellule on compte le nombre de voisins et on change l'état de la cellule en fonction de ce nombre

-reInit () : réinitialiser la grille

Pour le modèle de **Modèle de Schelling** on ajoute une méthode ajoutCellulesVacantes() qui parcourt la grille et ajoute toutes les cellules vacantes à la liste.

GrilleSimulator.java : cette classe implémente gui.Simulable en faisant un Override sur next() et restart().

Event.java : Génération d'un évènement en définissant une date.

GrilleEvent.java : Permet de faire un traitement sur les cellules lorsqu'un évènement est généré.

EventManager.java : Effectuer le managing des évènements et incrémenter la date à chaque translation des balles.

2) Tests effectués :

Pour chaque jeu , on initialise la grille avec des cellules dans état aléatoire (une couleur aléatoire pour le **Modèle de Schelling**), avec des dimensions de la grille bien spécifiques. Les tests du jeu de la vie et du jeu d'immigration montrent l'évolution de la grille à chaque date (mort et vie des cellules , immigration des cellules d'un état à un autre) , pour le modèle de Schelling on fixe le seuil des voisins, le simulateur montre la ségrégation des couleurs, on obtient la ségrégation à partir du **seuil K=2**. On peut changer le seuil pour voir la différence en modifiant la valeur de K dans le fichier **GrilleEvent.java.** Il faut uiliser le MAKEFILE pour tous les effectuer tous les tests.

Pour tout test, compile avec 'make' et exécute avec 'make exeTestSimulator'.

III)- Un modèle d'essaims : Les boids.

1)- Description des fichiers java :

Vecteur.java : Nous avons redéfini chaque coordonnée hérite de point package à un vecteur avec des attributs comme la vitesse et l'accélération. Il existe deux type de constructeurs pour la classe Vecteur car on a utilisé la même classe pour gérer les forces (séparation, cohésion et alignement), donc un vecteur de force n'a pas besoin de définir sa vitesse et accélération. Dans notre système, on suppose que la masse =1, donc force=accélération. Dans ce fichier, définir les fonctions séparation, cohésion et alignement.

Boids.java : Ce fichier permet de générer des points qui se déplacent librement dans l'espace comme un groupe d'oiseaux ou de poisson (boids) et contient essentiellement les méthodes suivantes :

-ajoutePoint() : Créer d'une liste de vecteur définit par les coordonnées aléatoires, la taille et la couleur de la balle et définir une vitesse et un accélération au point.

-behaviour() : Prend en paramètre un vecteur b, un vecteur cible (target) et une liste de vecteurs (boids) et permet de faire interagir le point avec les autres points du boids en utilisant les méthodes définis dans le fichier Vecteur.java (qui hérite le package « java.awt.Point ») : seperate() qui permet de séparer les agents très proches, cohesion() pour faire diriger un agent vers la positons moyenne du groupe de balles qui est le centre de masse et enfin la méthode align() qui consiste en le déplacement de l'agent dans la même direction que ses voisins.

-translate() : Définir un limite de la force appliqué sur l'agent pour ne pas avoir un mouvement trop rapide de la balle à l'aide de la méthode limit() et puis faire la translation après avoir fait les tests sur les dépassement de la taille de la fenêtre (comme dans la partie I).

-reInit() et toString (Idem. que la la partie I).

BoidsSimulator.java : ça implémente gui.Simulable en faisant un Override sur next() et restart().

Event.java : Génération d'un évènement en définissant une date.

BoidsEvent.java : Permet de faire un traitement sur les balles lorsqu'un évènement est généré.

EventManager.java : Effectuer le managing des évènements et incrémenter la date à chaque translation des balles.

2) Tests effectués :

On effectue les tests sur 3 parties différentes :

En premier lieu on fait le test sur un système de boids sans évènement => Les individus dans le boid essaie de se groupent, s'alignent et se séparent dès que un individu touche l'autre.

Après on fait un test sur un système de boids avec évènements =>Translation de boid à chaque 3 temps défini par gui passé.

À la fin on va appliquer à la fin ce qu'on a vu sur plusieurs groupes de boids qui ont des comportements différents (**la cohabitation**) :
i) un groupe de Lion : 3 individus essaient de séparer de l'un à l'autre le plus loin possible
ii) un groupe de Loup : 5 individus se groupent et restent dans une bonne distance
iii) Un groupe de Zèbre : une bonne quantité des individus essaient de se groupent et restent le plus proche possible

Veuillez changer différents comportements des boids dans fichier Lion.java, Loup.java, Zebre.java ou bien Boids.java.
Pour tout test, compile avec 'make' et exécute 'make exeGUI '.