TPL POO Livrable

Les Balles

Choix de conception

Classe Balls

- La classe Balls contient trois tableaux qui stockent la position des balles ainsi que leurs dynamiques.
- Le constructeur initialise le premier tableau balls[] qui contient les positions des balles courantes a des coordonnées fixées au centre puis les copie dans ballsOri[].
- Translate permet de déplacer toutes les balles de la simulation.
- reInit() réinitialise balls[] aux valeurs conservées dans ballsOri[].
- getNbBalls renvoie simplement la longueur du tableau balls ce qui évite d'ajouter une variable nbBalls dans la classe.
- getBall() permet d'accéder a une balle dans le tableau des balles courantes balls[].
- toString() renvoie un StringBuffer avec le nombre de points et leurs coordonnées.

BallsSimulator

- Le constructeur initialise le GUISimulator, les balles et les dessine grâce à la méthode drawBalls().
- Comme BallsSimulator réalise l'interface simulables elle doit redéfinir les méthodes next et restart.

Tests et Résultats

Au lancement de la simulation l'utilisateur précise le nombre de balles à simuler, toutes les balles seront initialement à la même position (au centre). En revanche, leurs vitesses et leurs directions seront aléatoirement définies.
Quand une balle rencontre une bordure, le vecteur vitesse représenté par dx et dy est modifié pour permettre à la balle de rebondir.

Automates Cellulaires

Les trois automates cellulaires héritent de la classe AutomateCellulaire et leurs simulations sont réalisées grâce aux classes qui héritent de AutomateSimulator.

Choix de conception

Cellule

La classe Cellule est utilisée indifféremment pour les trois automates. Celle-ci hérite de la classe Point à laquelle on ajoute un attribut entier représentant l'état de la cellule et les méthodes suivantes :

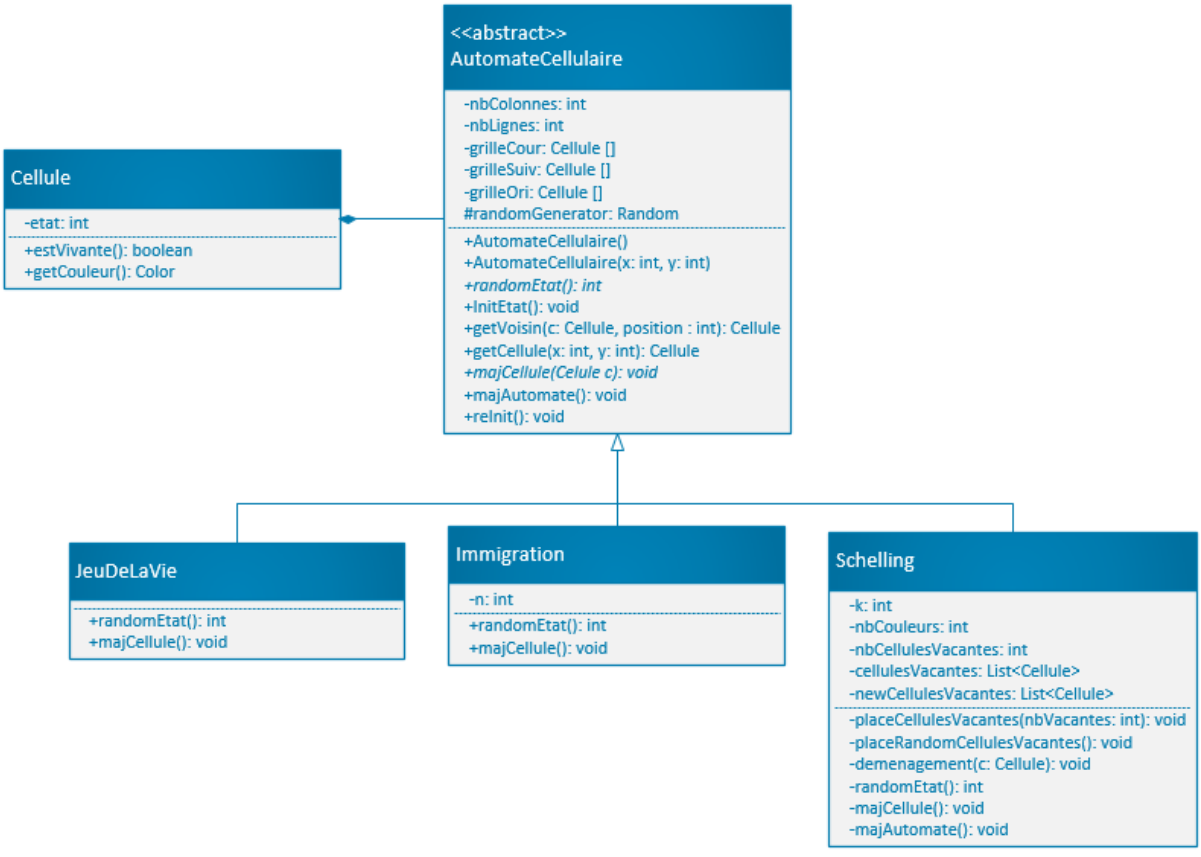
- getCouleur() qui renvoie selon l'état de la cellule une couleur parmi les 10 choisies
- estVivante qui renvoie une boolean true si l'état est supérieur à 1.

Classe abstraite AutomateCellulaire

Les automates héritent tous de la classe abstraite 'AutomateCellulaire'. Chaque automate possède trois grilles de cellules :

- grilleCour les cellules à l' instant t ;
- grilleSuiv les cellules à l'instant t+1, qui permet de faire les calculs ;
- grilleOri qui permet de sauvegarder les cellules dans leur état d'origine.
- Elle possède deux constructeurs : celui par défaut et un constructeur avec choix du nombre de lignes et de colonnes.
- Une méthode initEtat qui initialise les états lors de la création du jeu.
- Un abstract int randomEtat qui génère un état aléatoire selon l'automate.
- getVoisin(Cellule c, int i) qui permet de récupère la cellule voisine de c en position i.
- majAutomate() qui met à jour la grille grâce à la méthode abstraite majCellule qui elle-même met à jour une cellule selon les règles de chaque jeu.
  Lors d'un pas de simulation, les résultats des calculs sont toqués dans la grille suivante. Lorsque les calculs sont terminés pour ce step, on inverse les grilles suivante et courante.

Ainsi dans chaque automate il n'y a plus qu'à définir randomEtat, majCellule, leurs constructeurs, ainsi que leurs variables propres :



Immigration

Dans cette automate, on ajoute l'attribut n qui définit le nombre d'états.

Schelling

Dans cette automate, nous avons ajouté les attributs suivants :

- k : Seuil pour lequel une famille déménage ;
- nbCouleurs : Nombre de population différentes ;

- nbCellulesVacantes : Nombre de maison vide dans l'automate ;
- List cellulesVacantes : La liste des cellules libre au début d'un step ;
- List newCellulesVacantes : Liste des cellules vide à la fin du step.
  A chaque pas de simulation, on teste chaque famille pour voir si le nombre de voisin différents ne dépasse pas k. Si oui, on cherche dans la liste CelluleVacante une place libre et la famille déménage à cet emplacement. La place ainsi libéré est insérée dans la liste NewCelluleVacante.
  A la fin du step de simulation, les nouvelles cellules vacantes sont ajoutées à la liste des cellules vacantes, c'est pourquoi on redéfini la méthode majAutomate dans cette version.

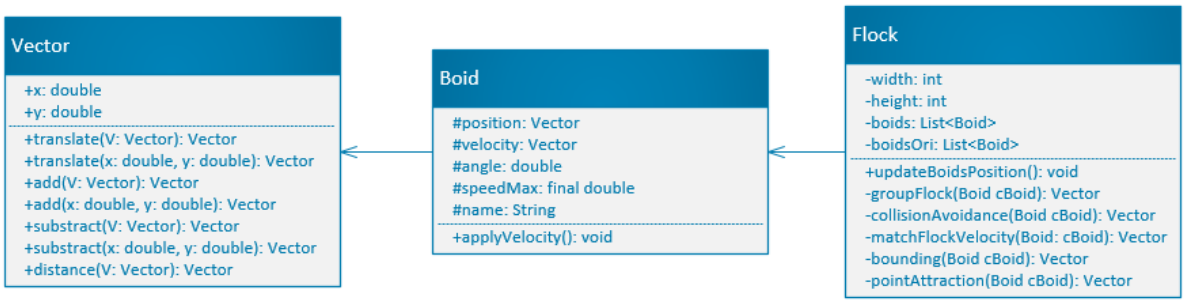
AutomateSimulator

AutomateSimulator implémente toute les méthodes utiles pour la simulation des automates. La classe mère utilise le concept de généricité. Une variable temporaire Automate est utilisée dans AutomateSimulator et permet de travailler indifféremment sur un des trois automate. Cela nous permet de factoriser toutes les méthodes pour les trois automates, seuls les constructeurs sont implémentés dans les fils afin de configurer l'automate.

Tests & Résultats

Pour tester les trois automates cellulaires il faut lancer les classes TestJdlvSimulator, TestImigrationSimulator et TestSchellingSimulator. Des scanners sont utilisés pour permettre à l'utilisateur de choisir la taille de la grille, et le nombre d'états/couleurs pour les deux derniers automates.

Boids



Choix de conception

Nous avons pu récupérer un squelette de code pour les boids sur un projet github hébergé à cette adresse : https://github.com/ShaunPlummer/Boids

Nous avons quand même dû apporter de nombreuses modification sur le code, nous l'avons adapté à notre simulateur graphique, corriger un ou deux bug dans la classe boid et implémenté quelques améliorations.

Cinq règles ont été implémentées pour simuler un troupeau, pour chacune d'elle, un ou deux paramètres est associés :

1. Déplacement vers le centre du groupe
   a. groupCenterFactor ;
2. Séparation entre les boids ;
   a. seperationFactor,
   b. seperationDistance ;
3. Déplacement dans la même direction que le groupe ;
4. Eviter les bords de la simulation ;
   a. boundingFactor,
   b. boundingDistance ;

5. Attraction vers la nourriture ;
    a. attractionFactor,
    b. Vector food.

Pour chaque règle un vecteur est calculé puis on divise le résultat par un facteur associé à la règle. L'ensemble de ces facteurs définit le comportement du flock.

Lors d'un pas de simulation, les règles génèrent cinq vecteurs qui sont alors sommés au vecteur velocity du boids. Un filtre peut être utilisé (coefficient coefFiltreVelocity) pour réduire ou augmenter l'influence de la vélocité courante du boid.

Résultats

Les résultats peuvent-être observés en exécutant la classe TestFlockSimulator. Deux flocks avec des comportements différents évoluent dans cette espace. On peut observer l'effet de chaque règle sur le troupeau.

Gestion d'évènements

Le package « évènements » inclut 4 classes :

- Event une classe abstraite décrivant un évènement. Un Event possède un attribut date qui permet d'établir la relation d'ordre (méthode compareTo) pour la PriorityQueue du manager et une méthode execute qui réalise l'évènement et ajoute le prochain au manager pour que nos animations « durent ».
- EventManager est le gestionnaire d'évènements, il possède un attribut EventQueue qui est une PriorityQueue (elle sert à stocker les évènements à réaliser tries selon leur date de réalisation) et un long currentDate qui lui permet de savoir quels évènements il reste à traiter avant la date courante.
- MessageEvent et TesteEventManager permettent de tester le gestionnaire.

La gestion d'évènements a ensuite été mise en place pour la simulation des balles, celles d'une flock, et celle des deux flocks proies/predateurs.

Deux Flocks : Proies / Predateurs

Comportement différent

Pour implémenter deux flocks aux comportements différents il nous a suffi de créer un nouveau simulateur PredateursProiesSimlateurs qui prend deux flocks « predateurs » et « proies » en attributs. Le constructeur appelle dans TestPPSimulator prend en paramètre le coefficient de groupe ce qui permet d'avoir des predateurs moins groupes que les proies.

Dessin

Ces flocks sont dessinées de façon différente dans drawFlock() une méthode de la classe PredateursProies qui dessine les deux flocks (Larges ronds rouges pour les predateurs et petits ronds bleus pour les proies).

Gestion d'évènements

Pour que les proies puissent se déplacer plus vite que les predateurs les évènements MajPredateurs ont lieu tous les deux pas de temps alors que MajProies a lieu tous les pas de temps. Cela se voit dans la création d'évènements a la volée dans les méthodes execute : Chaque exécution de MajPredateurs ajoute un nouvel évènement MajPredateurs a la date t+2.