Systèmes Multi-agents

Simulation

Claude Moulin

Université de Technologie de Compiègne

IA04

Sommaire

Domaines d'utilisation de simulation à base d'agents

- Biologie : comportement et organisation de diverses espèces animales (fourmis, singes, boeufs, poissons)
- Ecologie : gestion de ressources renouvelables sous l'influence de l'homme (pêche, forêt)
- Médecine : Modèles épidémiologiques (grippe, sras)
- Socio-ethnologie : migration de population, changement d'opinion
- Urbanisme - Architecture : analyse du développement des villes, de la gestion des déplacements des piétons dans des aéroports, des gares, déplacements des véhicules, etc.

Exemples

- Site :
  `http://www.massivesoftware.com/index.html`
- Visualisation réaliste d'animations basées sur des agents autonomes :
  comportement de foules avec simulation d'émotions (bravoure, fatigue, joie, etc .)

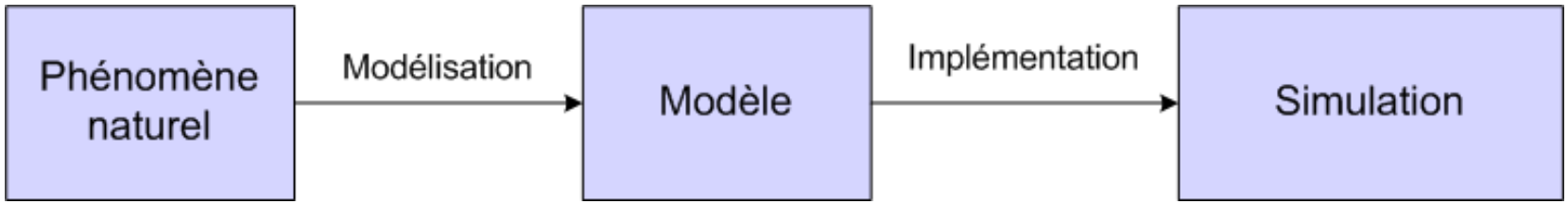
Types de réalisations

- Films
- Télévision, jeux
- Education
- Architecture
- Mécanique

Sommaire

Sommaire

Cadre d'une simulation



Modèle : définition

- Définition de Marvin Minsky, 1965
  To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A

Types de simulation

- Simulation continue
- Simulation discrète
- Simulation discrète par agents

Simulation continue

- Le modèle du système se présente sous la forme d'équations différentielles à résoudre.
- Résolution analytique du problème.
- Résultat au niveau macroscopique : courbe, surface représentant une fonction étudiée.

Simulation discrète

- Le modèle du système est soumis à une succession d'évènements qui le modifient.
- Appliquée à des systèmes de grande taille.
- Deux grandes catégories :
  - modèle asynchrone ou time-slicing : on simule à chaque fois le passage d'une unité de temps sur tout le système.
  - modèle synchrone ou event-sequencing : on calcule l'arrivée du prochain événement qui est propagé dans le modèle
- Simulations rapides, bien qu'un peu plus complexes à programmer.

Simulation par agents

- Modèles centrés sur des entités et leurs interactions.
- La dynamique générale du système est issue des interactions entre ces entités.
- Fonctionnement asynchrone (time-slicing).
- Définition de Système multi-agents : population d'agents autonomes en interaction

Limites des modèles numériques

- Lorsque le système possède un très grand nombre de paramètres, le modèle d'équations est parfois difficile à résoudre et à interpréter.
- Difficulté ou même impossibilité de représenter les comportements individuels.
- Ne représentent pas les comportements mais les conséquences des comportements.
- Ne permet pas de comprendre les structures spatio-temporelles et sociales (bancs de poissons, fourmis).

Objectifs de la Simulation par agents

- Avantage : permet de tester rapidement une hypothèse.
- Inférer sur la nature du fonctionnement des entités d'un système complexe.
- Mettre en évidence les aspects émergents d'un phénomène dépendant de choix individuels.
- Mettre en évidence l'existence ou non de situations stables.

Problèmes de la Simulation par agents - 1

- Gestion d'échelles de temps et d'espace hétérogènes.
  Comment intégrer des processus qui se déroulent à des échelles de temps et d'espace très hétérogènes ?
- Gestion des paramètres.
  Comment gérer un grand nombre de paramètres ?
  Comment vérifier l'influence relative des paramètres ?

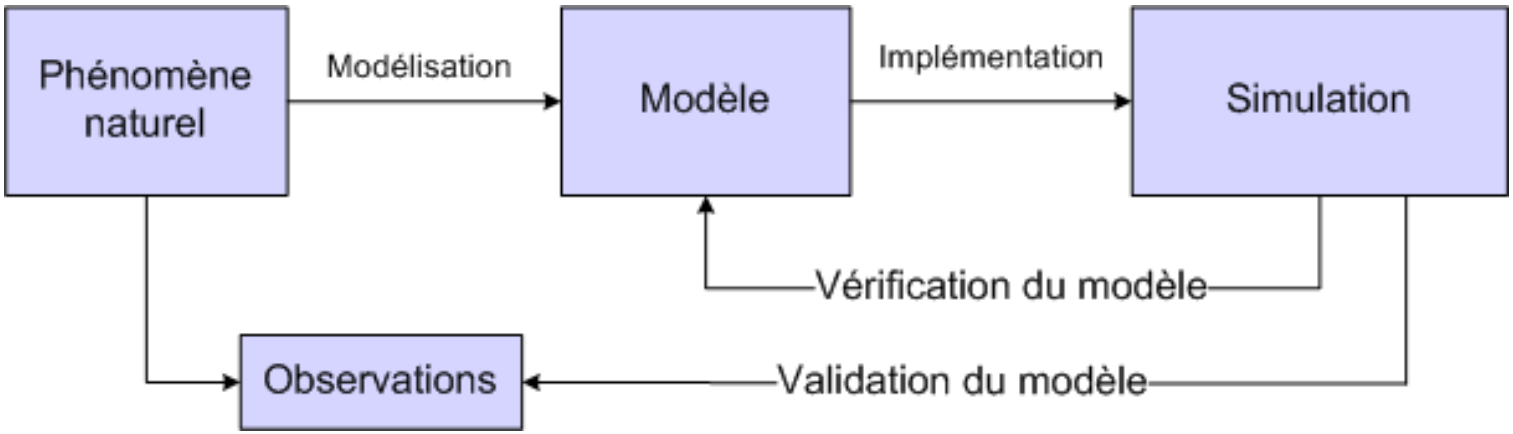
Problèmes de la Simulation par agents - 2

- Fiabilité des systèmes multi-agents.
  Fiabilité = assurance que les sorties des simulations découlent uniquement des mécanismes que l'on pense avoir élaborés dans le modèle (danger de biais).
- Réplicabilité et lisibilité des systèmes multi-agents.
- Comparaison entre modèles

Problèmes liés à la Simulation

- Validation des modèles .
  Validation qualitative ? il n'existe pas de validation parfaite, encore moins définitive.
- Danger de prise de décision selon un modèle.

Cadre d'une simulation



Sommaire

Modélisation d'un phénomène - 1

- Décomposition du phénomène en un ensemble d'éléments discrets autonomes dont les interactions reproduisent le phénomène : vision déjà distribuée du phénomène à modéliser.
- Phase semblable au processus de réification intervenant dans les méthodes de conception orientées objet.

Modélisation d'un phénomène - 2

- Modélisation par un agent de chacun des éléments identifiés.
- Choix nécessaire pour définir les connaissances des agents, leurs capacités fonctionnelles, leurs comportements et les modes d'interaction à l'encontre des autres agents.
- Le niveau de granularité de la modélisation doit se placer en accord avec le domaine d'application choisi.

Modélisation d'un phénomène - 3

- Définition de l'environnement des agents : espace dans lequel évoluent les agents et des lois qui le gouvernent.
- La définition permet d'affiner la description des actions possibles des agents, ainsi que les moyens de communication.
- Définition des objets inertes : éléments qui ne sont dotés d'aucune capacité d'action ni de communication.

Modélisation d'une population - 1

Modélisation centrée individus.

- Population d'individus quelconques
  - éthologie : insectes, fourmis, etc.
  - sociologie : personnes
- Modélisation de leur structure interne (physique, mentale).
- Modélisation de leur comportement (réactif, cognitif).
- On associe un agent à chaque individu de la population .

Modélisation d'une population - 2

- Implémentation des lois du monde qui est simulé.
- Simulation généralement discrète :
  - Le temps est simulé
  - A chaque pas de temps, le système multi-agent évolue.

Visualisation d'une forme émergente

- Montrer les entités qui sont sensées émerger de la population :
  - Formes : attracteurs, fractales,
  - Structures : organisations, prise de pouvoir,
  - Comportements des individus.

Environnement

- Un agent est situé en dans un espace 2D ou 3D et il possède un voisinage :
  - Il peut le percevoir localement et non globalement
  - Il peut se le représenter
- Il peut agir sur cet environnement
  - Relativement, indirectement : par exemple en se déplaçant
  - Absolument : par modification de l'état physique ou mental des éléments du voisinage (objets, autres agents)

Agents

- Plusieurs modélisaton en fonction de théorie scientifiques.
- Exemple de modélisation en éthologie :
  - Le comportement d'un animal peut être caractérisé par un ensemble de tâches indépendantes, composées d'une séquence de comportements moteurs élémentaires appelés primitives (éléments innés).
  - Chacune de ces tâches est exclusive.
  - Le déclenchement d'une tâche provient d'une stimulation externe ou interne, concrétisée par des stimuli de force variable, combinée à une motivation préexistante.
  - La motivation s'exprime à l'aide d'un seuil inhibiteur et d'un poids (expérience antérieure de l'animal).

Trois concepts fondamentaux

- Modélisation
- Simulation
- Visualisation

Eléments d'une simulation

- Modèle : partie non visible contenant l'information du système, la logique manipulant cette information
  - Environnement
  - Agents
- Moteur de la simulation
- Visualisation : ensemble de fenêtres permettant de visualiser certains éléments du modèle et d'interagir éventuellement avec eux.

Moteur

- Responsable de la notion de temps
- Boucle où chaque itération correspond à un pas d'horloge.
- A chaque itération, les agents actifs interagissent avec l'environnement.
- Des agents peuvent devenir inactifs, d'autres naître ou disparaître en cours de simulation, etc.
- Il est souhaitable de pouvoir définir des points de contrôle :
  - arrêter le temps
  - figer le modèle
  - relancer le moteur
  - reprendre le modèle dans l'état où il a été figé.

Exemples de plateformes

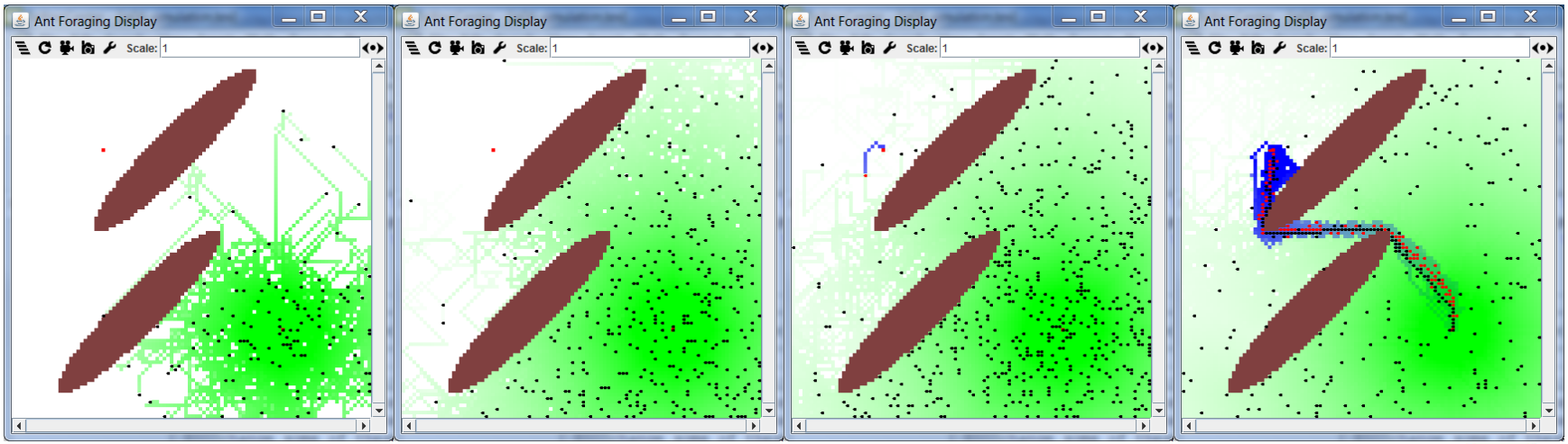
- MASON : Multi-Agent Simulator Of Neighborhoods, ou Networks
  `http://cs.gmu.edu/~eclab/projects/mason/`
- Swarm : `www.swarm.org`
- RePast : `repast.sourceforge.net/`
- Massive Software (commercial) : `www.massivesoftware.com`
- Cormas : `cormas.cirad.fr`
- NetLogo : `http://ccl.northwestern.edu/netlogo/`
  StarLogo : `http://education.mit.edu/starlogo/`
- TurtleKit (MadKit) : `http://www.turtlekit.org/`

Sommaire

Sommaire

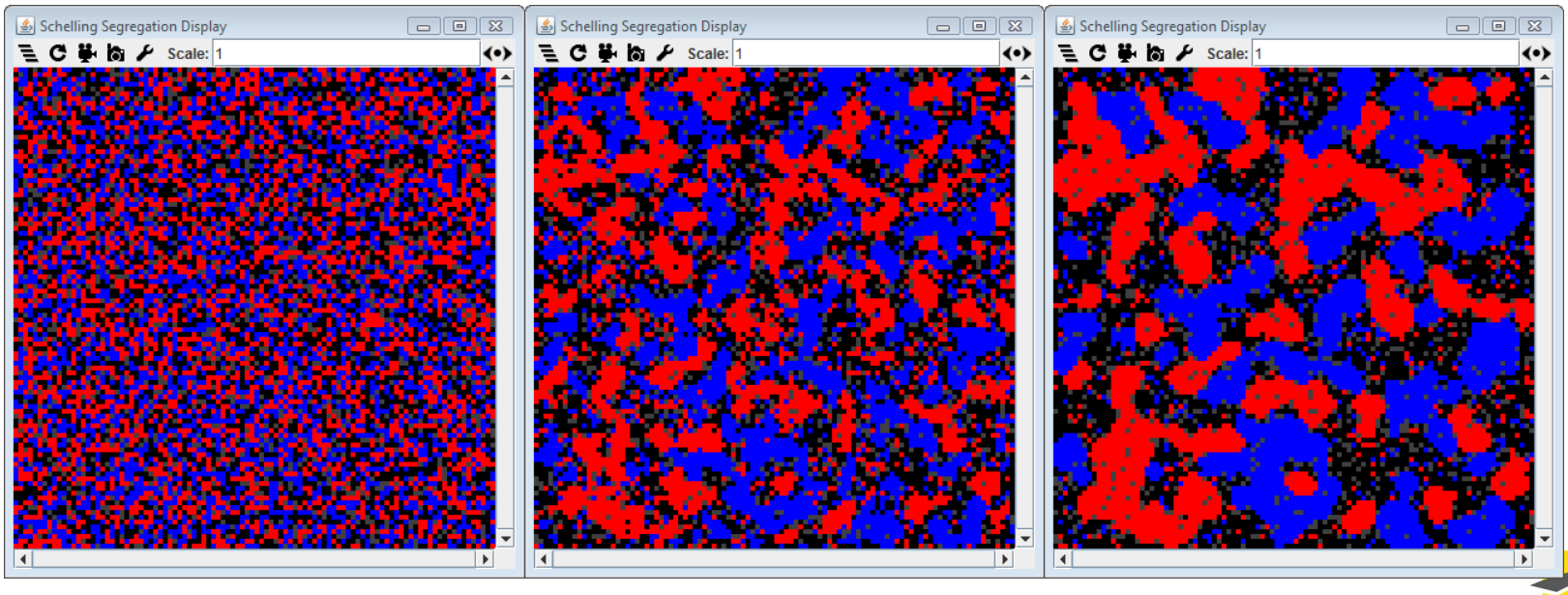
Ant Foraging

Simulation of artificial ants foraging from a nest, discovering a food source in the face of obstacles, and then establishing a trail between the nest and food source. The model uses two pheromones which set up gradients to the nest and to the food source respectively. The pheromones evaporate as well.



Thomas Schelling's Segregation Model

A variation of Thomas Schelling's Segregation Model. We model two groups of people : Red and Blue. Each person prefers that there be at least threshold number of like-colored people living within neighborhood distance away from him. If this is not the case, he will get up and move to another location.



Sommaire

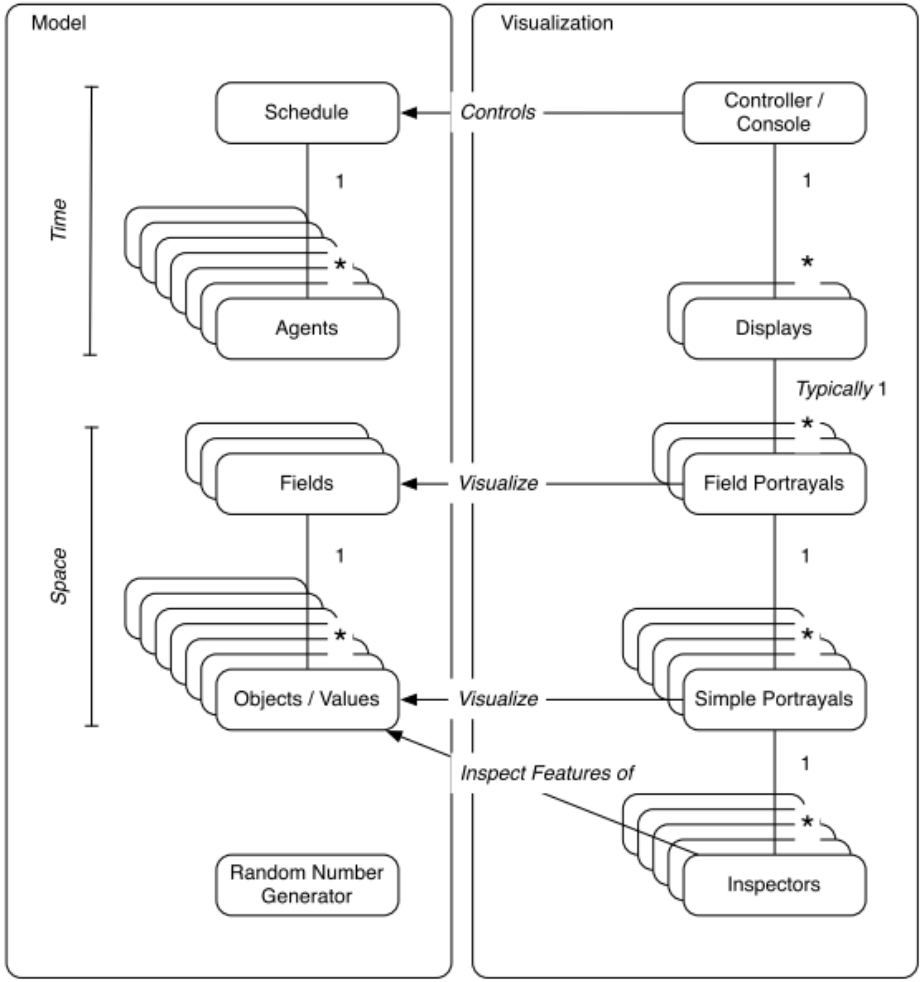
MASON

- Multi-Agent Simulator Of Neighborhoods, ou Networks
- Site :
  `http://cs.gmu.edu/~eclab/projects/mason/`

Architecture

- MASON comprend deux parties bien séparées
  - Le modèle (la simulation proprement dite)
  - La visualisation (2D ou 3D)
- Avantages :
  - Le modèle peut s'exécuter sans visualisation
  - Le modèle peut avoir plusieurs visualisations
- Caractéristiques :
  - On peut mettre des points de contrôle dans le modèle
  - Il peut être stoppé et sauvegardé sur disque
  - Il peut être repris même sur un autre ordinateur

Architecture



Modèle

- Un modèle MASON est un objet d'une classe héritant de `sim.engine.SimState`.
- Il possède une représentation du temps (de l'écoulement du temps). C'est une planification d'événements discrets où différents agents peuvent être appelés à exécuter certaines actions.
- Il possède une représentation de l'espace (l'environnement).
- Un générateur de nombres aléatoires de haute qualité permet que l'ordre dans lequel les agents sont appelés à chaque étape soit véritablement aléatoire et ne possède pas de biais.

Visualisation

- Dispositifs de visualisation 2D et 3D (fenêtres).
- Possibilités de plug-in : SIG
- Une visualisation est un objet d'une classe héritant de `asim.display.GUIState`.
- Il contient un contrôleur (une console) chargé de gérer la planification du temps (démarrer, stopper, etc.)

Modèle - vue

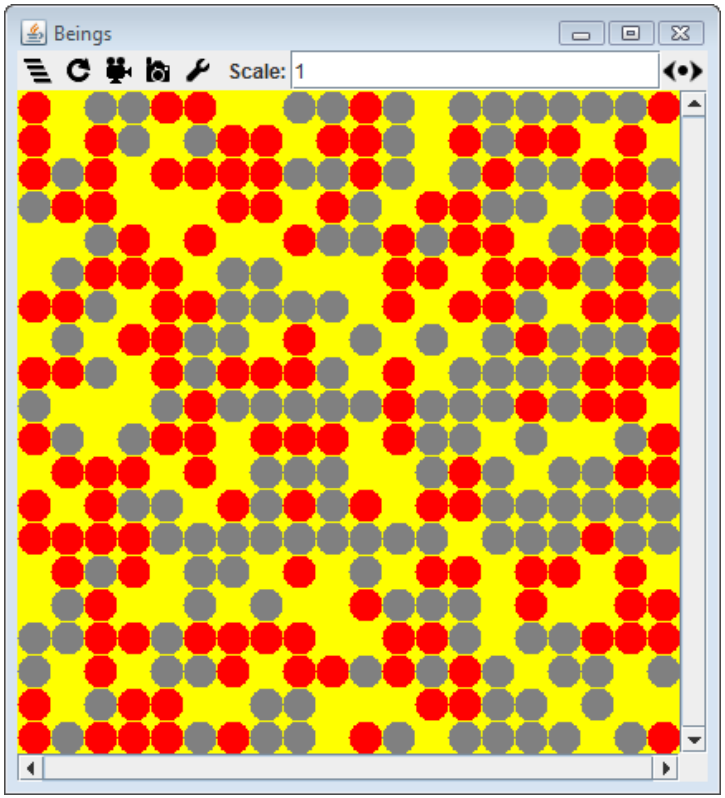
- Dans le modèle, l'environnement contient des champs : structure de données relatives à des objets ou des valeurs.
  - Réseau, espace continu de points, grille de cellules.
- La vue (fenêtre) permet de représenter les champs (field portrayal). Dans ces représentations de champs sont représentés des objets ou des valeurs.
- Il est possible d'inspecter ces objets ou ces valeurs.

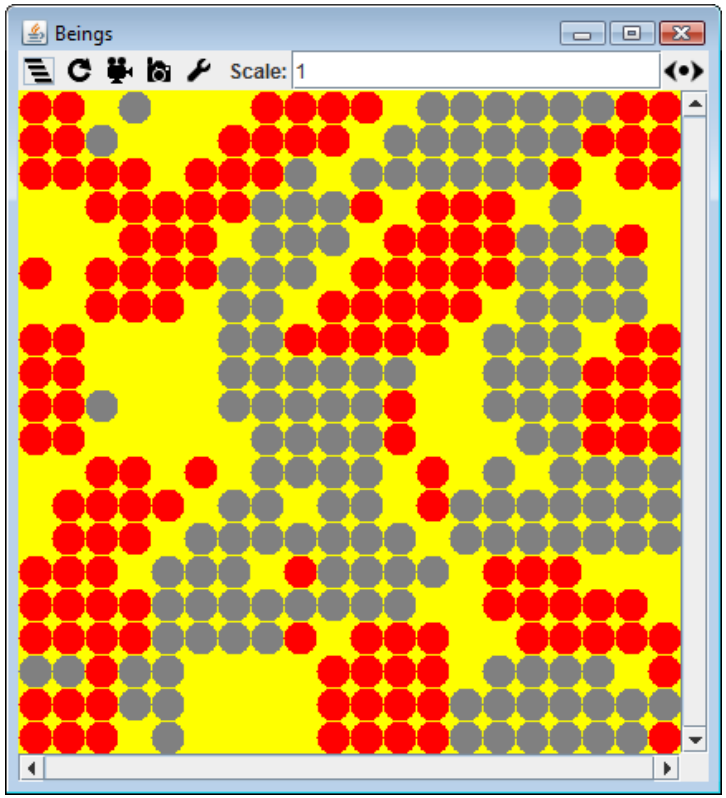
Sommaire

Principe

- Placer sur une grille de n x n des individus de deux types A et B.
  Appliquer le principe de Schelling
- Etudier l'hypothèse du déplacement en fonction des règles :
  - Un individu ne bouge pas si le taux de voisins semblables >= 1/3
  - Un individu bouge et cherche un endroit meilleur si le taux de voisins semblables < 1/3

Vue





Méthode

- Créer le modèle.
- Créer les classes agents.
- Créer la visualisation.
- Spécialiser les agents.

Classes

- La classe modèle ; elle hérite de `sim.engine.SimState`.
- La classe visualisation ; elle hérite de `sim.display.GUIState`.
- La classe Main pour lancer la simulation
- Les classes agents ; elles implémentent `sim.engine.Steppable` ;
  les agents sont activés à chaque itération (méthode `step(SimState)`

Modèle

```
public class Beings extends SimState {
  public Beings(long seed) {
   super(seed);
  }
}
```

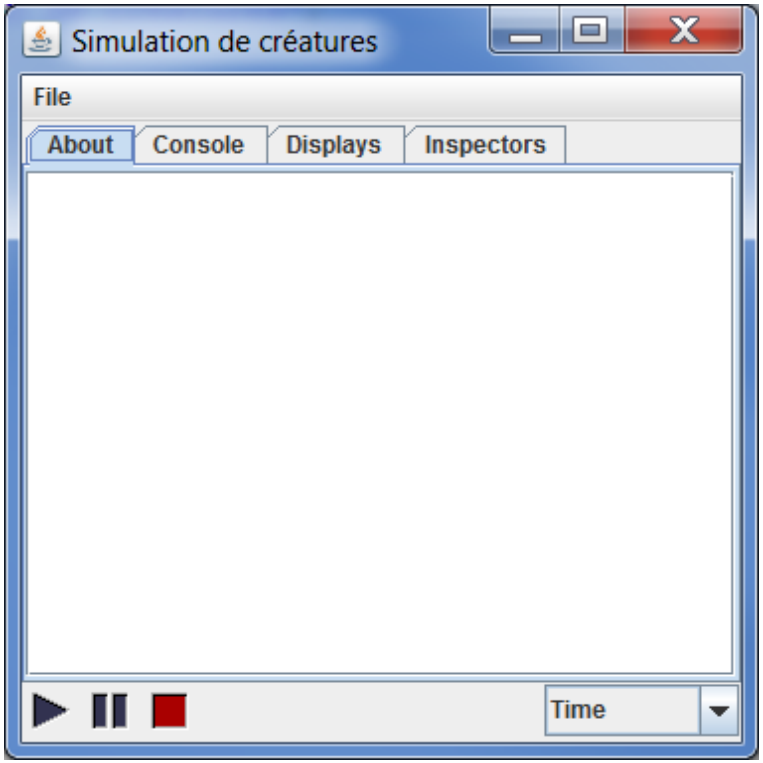
Visualisation

```
public class BeingsWithUI extends GUIState {
  public BeingsWithUI(SimState state) {
    super(state);
  }
  public static String getName() {
   return "Simulation de créatures";
  }
}
```

Main

```
public class BeingsMain {
 public static void main(String[] args) {
  runUI();
 }
 public static void runUI() {
  Beings model = new Beings(System.currentTimeMillis());
  BeingsWithUI gui = new BeingsWithUI(model);
  Console console = new Console(gui);
  console.setVisible(true);
 }
}
```

Console



Le bouton Play lance la simulation : start() sur le SimState(), suivis par des appels répétés au contrôleur.

Modèle

- Environnement, une grille de n x n : `ObjectGrid2D`
- Des agents (classes implémentant `sim.engine.Steppable`) de type A et B placés initialement sur la grille

Modèle

```
public class Beings extends SimState {
 public static int GRID_SIZE = 20;
 public static int NUM_A = 150;
 public static int NUM_B = 150;
 public ObjectGrid2D yard =
    new ObjectGrid2D(GRID_SIZE,GRID_SIZE);
 ...
 public Beings(long seed) {
  super(seed);
 }
 public void start() {
  super.start(); yard.clear();
  // Positionnement des agents de type A et B
 }
}
```

Modèle - positionnement

```
List<Boolean> freeLocations =
    IntStream
    .range(0, GRID_SIZE*GRID_SIZE)
    .mapToObj(p -> true)
    .collect(toList());
Function<Integer,Int2D> locationFromRow =
  row -> new Int2D(row / GRID_SIZE,
                   row % GRID_SIZE);
Predicate<Integer> isFreeLocation =
  row -> freeLocations.get(row);
```

Modèle

```
public class Beings extends SimState {
 ...
 public void start() {
  super.start();
  yard.clear();
  addAgentsA();
  addAgentsB();
  }
...
}
```

Placement des agents - 1

```
private void addAgentsA() {
 for(int i = 0; i < NUM_A; i++) {
   TypeA a = new TypeA();
   Int2D location = getFreeLocation();
   yard.set(location.x,location.y,a);
   a.x = location.x;
   a.y = location.y;
   schedule.scheduleRepeating(a);
 }
}
```

Placement des agents - 2

```
private Int2D getFreeLocation() {
 Long freeSize = freeLocations.stream()
   .filter(place -> place == true)
   .collect(counting());
 Long placeForAgent = random.nextLong(freeSize);
 int place = 0; int position = -1;
 while (place <= placeForAgent) {
  position++;
  if (isFreeLocation.test(position)) {
   place++ ;
  }
 }
 Int2D p = locationFromRow.apply(position);
 return p;
}
```

Visualisation

- Objet de visualization 2d (`sim.display.Display2D`), le display
  Le display peut afficher plusieurs champs.
- 1 champ (portrayal) de type : `ObjectGridPortrayal2D`.
- 1 fenêtre contient le display : `JFrame`
- Le portrayal dessine et inspecte les individus de la grille en appelant des portrayals simples destinés à représenter ces individus : de type A ou B

Visualisation - code

```
public class BeingsWithUI extends GUIState {
public static int FRAME_SIZE = 400;
 public Display2D display;
 public JFrame displayFrame;
 ObjectGridPortrayal2D yardPortrayal =
                    new ObjectGridPortrayal2D();
 public BeingsWithUI(SimState state) {
  super(state);
 }
 public void start() {
  super.start(); setupPortrayals();
 }
 public void load(SimState state) { ... }
 public void setupPortrayals() { ... }
 public void init(Controller c) { ... }
}
```

Initialisation

init est appelée lorsque la visualisation (`GUIState`) est créée.

```
public void init(Controller c) {
 super.init(c);
 display = new Display2D(FRAME_SIZE,FRAME_SIZE,this);
 display.setClipping(false);
 displayFrame = display.createFrame();
 displayFrame.setTitle("Beings");
 c.registerFrame(displayFrame);
 displayFrame.setVisible(true);
 display.attach( yardPortrayal, "Yard" );
}
```

Description des objets

```
public void setupPortrayals() {
 Beings beings = (Beings) state;
 yardPortrayal.setField(beings.yard );
 yardPortrayal.setPortrayalForClass(
    TypeA.class, getTypeAPortrayal());
 yardPortrayal.setPortrayalForClass(
    TypeB.class, getTypeBPortrayal());
 display.reset();
 display.setBackdrop(Color.yellow);
 display.repaint();
}
```

Visualisation des agents

```
private OvalPortrayal2D getTypeAPortrayal() {
 OvalPortrayal2D r = new OvalPortrayal2D();
 r.paint = Color.RED;
 r.filled = true;
 return r;
}
```

Spécialisation des agents

```
public class AgentType implements Steppable {
 public int x, y;
 @Override
 public void step(SimState state) {
  Beings beings = (Beings) state;
  if (friendsNum(beings) * 3 < 8) {
    move(beings);
  }
 }
 protected int friendsNum(Beings beings) {
  return friendsNum(beings,x,y);
 }
...
}
```

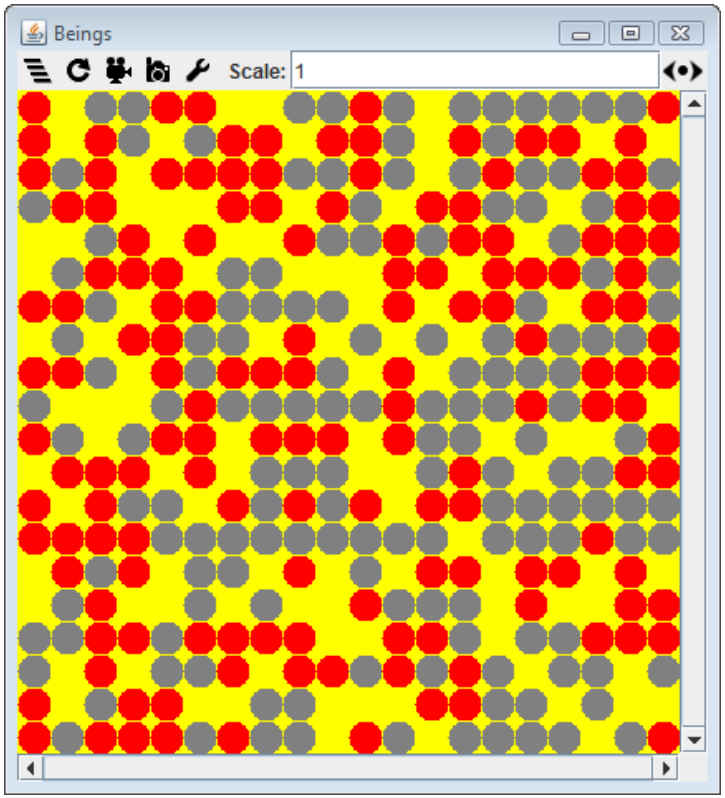
Comptage

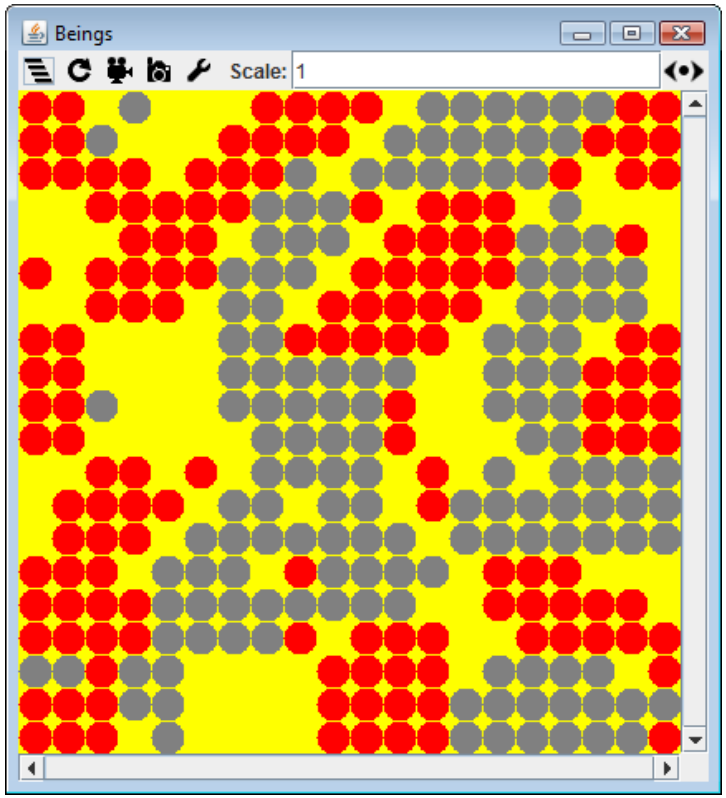
```
protected int friendsNum(Beings beings,int l,int c) {
 int nb = 0;
 for (int i = -1 ; i <= 1 ; i++) {
  for (int j = -1 ; j <= 1 ; j++) {
   if (i != 0 || j != 0) {
    Int2D flocation = new Int2D(beings.yard.stx(l + i),
                                beings.yard.sty(c + j));
    Object ag = beings.yard.get(flocation.x,flocation.y);
    if (ag != null && ag.getClass() == this.getClass())
      nb++;
   }
  }
 }
 return nb;
}
```

Mouvement

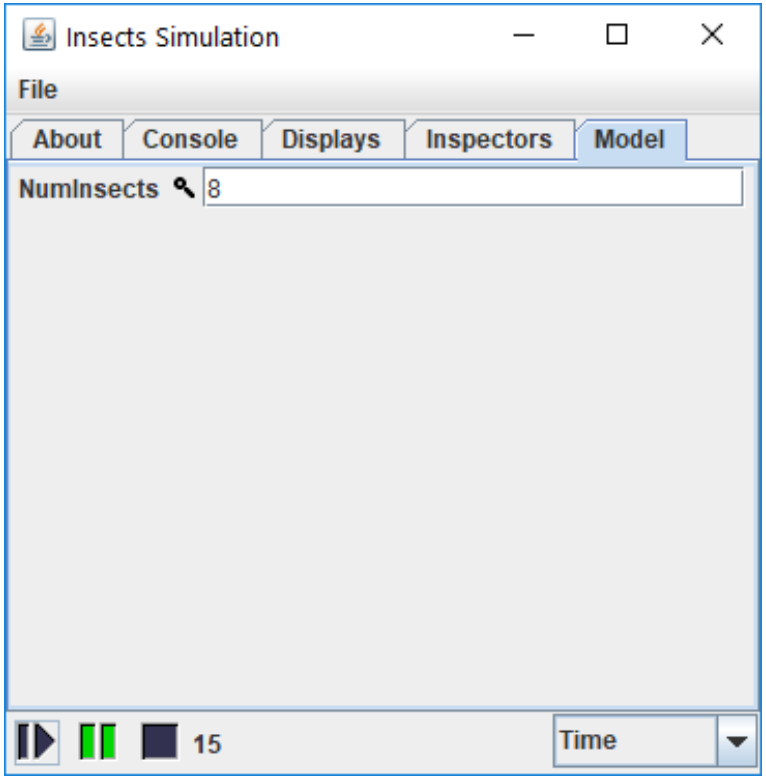
```
public void move(Beings beings) {
 int n = beings.random.nextInt(8);
 switch(n) {
 case 0:
    if (beings.free(x-1, y)) {
      beings.yard.set(x, y, null);
      beings.yard.set(beings.yard.stx(x-1), y, this);
      x = beings.yard.stx(x-1);
    }
    break;
   ... }
}
```

Résultat



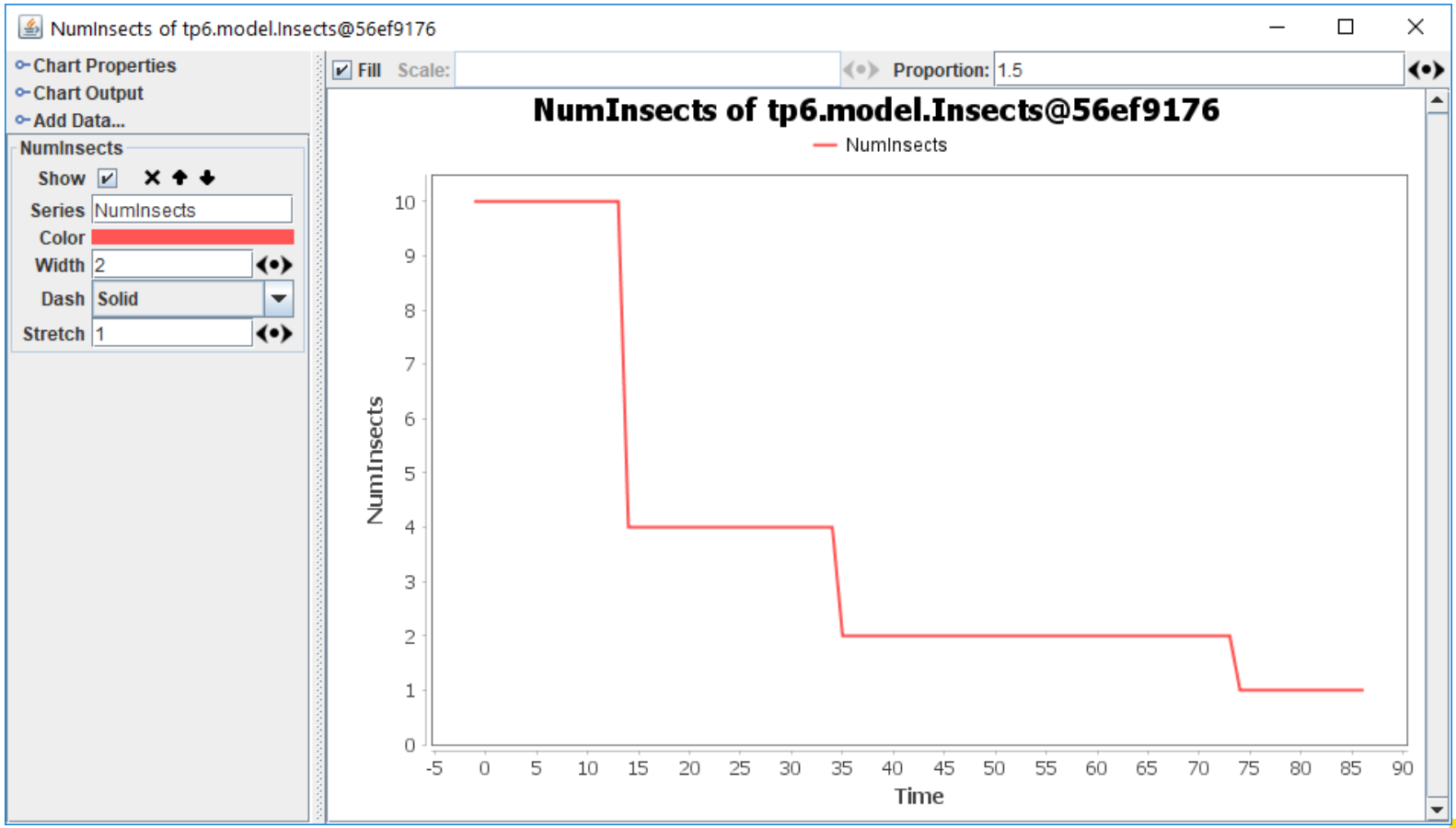


Caractéristiques : graphique



Cliquer sur la petite loupe à côté du champ pour faire apparaître le graphique.

Caractéristiques : graphique



Adaptation Portrayal

```
public class StrangeOvalPortrayal
          extends OvalPortrayal2D {
 public StrangeOvalPortrayal() {
  super();
  paint = Color.GRAY;
  filled = true;
}
@Override
 public void draw(Object object, Graphics2D graphics,
     DrawInfo2D info) {
  ...
 }
}
```

Adaptation Portrayal

```
public class StrangeOvalPortrayal
          extends OvalPortrayal2D {
 public StrangeOvalPortrayal() {
 ...
}
@Override
 public void draw(Object object, Graphics2D graphics,
     DrawInfo2D info) {
 AgentType agent = (AgentType) object;
 if (agent.x % 5 == 0 && agent.y % 5 == 0)
   scale = 2;
 else scale = 1;
 super.draw(object, graphics, info);
 }
}
```

Type de Yard

- SparseGrid2D : grille 2D avec plusieurs éléments par cellule (coordonnées entières)
- ObjectGrid2D : grille 2D avec 1 élément par cellule (coordonnées entières)
- Continuous2D : espace continu (coordonnées réelles)

SparseGrid

```
Int2D location = ...
yard.setObjectLocation(<ref element>,
                       location);

Int2D flocation = ...
Bag bag =
    state.yard.getObjectsAtLocation(flocation);
```

ObjectGrid

```
Int2D location = ...
state.yard.set(location.x, location.y,
                    <ref element>);

Int2D flocation = ...
Object object = state.yard.get(flocation.x,
                               flocation.y);
```

Continuous2D

```
public Continuous2D yard =
  new Continuous2D(discretization, width, height);

Double2D location = ...
state.yard.setObjectLocation(<ref element>,
                             location);

Double2D position = ...
double distance = ...
Bag neighbors =
 yard.getNeighborsExactlyWithinDistance(position,
                                        distance);
```